O'REILLY®

3-е издание

Arduino
БОЛЬШАЯ
книга рецептов

Майкл Марголис
Брайан Джепсон
Николас Роберт Уэлдин

bhv®

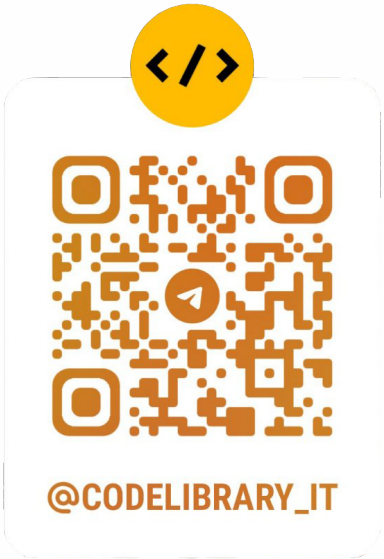
THIRD EDITION

Arduino Cookbook

Recipes to Begin, Expand, and Enhance Your Projects

Michael Margolis, Brian Jepson, and Nicholas Robert Weldin

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®


@CODELIBRARY_IT

Майкл Марголис
Брайан Джепсон
Николас Роберт Уэлдин

Arduino
БОЛЬШАЯ
книга рецептов

3-е издание

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.89
ББК 32.973.26-018
М25

Марголис, М.

М25 Arduino. Большая книга рецептов. — 3-е изд.: Пер. с англ. / М. Марголис, Б. Джепсон, Н. Р. Уэлдин. — СПб.: БХВ-Петербург, 2021. — 896 с.: ил.

ISBN 978-5-9775-6687-2

Этот справочник будет идеальным пособием для любого, кто хочет создавать проекты на базе популярной платформы Arduino. Он содержит подробное описание решений свыше 200 практических задач по созданию различных устройств и приспособлений, включая проекты для Интернета вещей, мониторинга окружающей среды, системы для определения местонахождения и положения в пространстве, а также устройств, которые реагируют на касание, звук, тепло и освещение. Все примеры третьего издания обновлены для версии 1.8 среды Arduino IDE с учетом современных концепций программирования. Каждое решение включает в себя программный код с подробными комментариями, его анализ и обсуждение возможных проблем. Книга будет полезна как начинающим, так и опытным разработчикам.

Для радиолюбителей

УДК 004.89
ББК 32.973.26-018

Группа подготовки издания:

Руководитель проекта	*Игорь Шишигин*
Зав. редакцией	*Людмила Гауль*
Компьютерная верстка	*Ольги Сергиенко*
Оформление обложки	*Карины Соловьевой*

Подписано в печать 31.05.21.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 72,24.
Тираж 2000 экз. Заказ № 1330.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано в ОАО "Можайский полиграфический комбинат",
143200, г. Можайск, ул. Мира, д. 93

ISBN 978-1-491-90352-0 (англ.)
ISBN 978-5-9775-6687-2 (рус.)

Оглавление

Предисловие

Майкл Марголис (Michael Margolis), Брайан Джепсон (Brian Jepson) и Ник Велдинг (Nick Welding) написали эту книгу, чтобы помочь вам увидеть все те захватывающие возможности, которые открывает перед вами использование платформы Arduino.

Платформа Arduino представляет собой комбинацию семейства *микроконтроллеров* (небольших компьютеров) и среды разработки программного обеспечения, позволяющей с легкостью создавать программы (на жаргоне Arduino называемые *скетчами*), которые могут взаимодействовать с окружающим миром. Устройства на основе Arduino способны ощущать прикосновения, звук, изменения расположения в пространстве, тепло и свет и реагировать на такие воздействия. Этот вид технологии, которая часто называется *физическими вычислениями*, используется в самых разнообразных устройствах: от смартфонов до автомобильных электронных систем. Платформа Arduino позволяет любому желающему, даже тому, кто не обладает опытом программирования или сборки электронных схем, использовать эту сложную, но богатую разнообразными возможностями технологию.

Для кого предназначена эта книга?

Эта книга найдет своего читателя среди тех, кто интересуется организацией взаимодействия компьютерных технологий с окружающей средой, кто хочет быстро реализовать аппаратное и программное решение возникших у него идей. Рассматриваемые в ней практические примеры позволят вам разобраться с широким кругом задач такого взаимодействия. Книга также содержит подробные сведения, которые помогут вам адаптировать приведенные в ней примеры под ваши конкретные нужды. В рамках книги не представляется возможным изложить общую теоретическую информацию по рассматриваемым в ней технологиям, поэтому вам следует черпать ее из внешних источников, ссылки на которые приводятся на всем протяжении изложения материала книги. В частности далее, в *разд. «Что не вошло в эту книгу?»*, вы найдете ссылки на книги, из которых те из вас, кто не имеет опыта в области программирования и/или электроники, смогут почерпнуть необходимую им общую информацию.

Если вы не обладаете опытом программирования, но хотите самостоятельно создать интерактивный проект, приведенные в этой книге примеры решения свыше двухсот популярных задач помогут вам научиться разрабатывать эффективный код. Для тех из вас, кто не имеет абсолютно никаких знаний ни в области электроники,

ни в области программирования, может оказаться полезным обратиться за консультацией к какой-либо книге для начинающих — например, к книге «Getting Started with Arduino» («Знакомство с Arduino») авторов Massimo Banzi (Массимо Банци) и Michael Shiloh (Майкл Шило), изданной сообществом Make (Make Community)[1].

Если же у вас есть общий опыт в области электроники и программирования, но вы не знакомы с платформой Arduino, наша книга поможет вам начать быстро создавать решения, демонстрируя, как применить конкретные возможности Arduino для ваших проектов.

Книга будет также полезна и тем, кто уже работает с Arduino, дав им возможность быстро овладеть новыми методами, которые рассматриваются на основе практических примеров. Показывая, как решать задачи, используя еще неизведанные для вас возможности, она поможет вам быстро перейти к реализации более сложных проектов.

Даже опытные программисты на языке C/C++ смогут найти в ней полезные примеры того, как использовать низкоуровневые ресурсы процессоров семейства AVR (прерывания, таймеры, шину I^2C, сеть Ethernet и т. п.) для создания приложений в среде разработки Arduino.

Организация книги

Книга содержит информацию, охватывающую широкий диапазон возможностей платформы Arduino: от базовых понятий и типичных задач до продвинутых технологий. Каждый метод объясняется с помощью примера, демонстрирующего реализацию конкретной возможности. Книгу не обязательно читать в строгой последовательности изложения материала. Если в примере используется метод, рассмотренный в другом примере, вместо повторного приведения всех подробностей этого метода просто дается ссылка на соответствующий пример.

- **Глава 1. Начало работы**.

  Введение в платформу Arduino и информация по установке и запуску среды разработки Arduino и соответствующих аппаратных средств. Здесь также дается краткий обзор некоторых наиболее популярных новых плат Arduino. В следующих двух главах рассматривается среда Arduino для разработки программного обеспечения.

- **Глава 2. Программирование на языке Arduino**.

  Основные понятия и задачи программирования.

- **Глава 3. Математические операции**.

  Наиболее часто используемые математические функции.

---

[1] К сожалению, эта книга не издана на русском языке, поэтому мы рекомендуем вам в качестве адекватной замены книгу Джереми Блума «Изучаем Arduino: инструменты и методы технического волшебства», 2-е изд. пер. с англ (**https://bhv.ru/product/izuchaem-arduino-instrumenty-i-metody-tehnicheskogo-volshebstva-2-e-izd-per-s-angl/**) — примечание от русской редакции, если не указано иное.

- **Глава 4. Последовательная связь**.

  Подключение платы Arduino к компьютеру и другим устройствам и взаимодействие с ними. Для ввода и вывода данных в Arduino наиболее часто используется последовательная связь стандарта RS-232, которая и задействована в большинстве примеров этой книги.

- **Глава 5. Простой ввод цифровых и аналоговых данных**.

  Основные способы считывания цифровых и аналоговых сигналов.

- **Глава 6. Считывание данных с датчиков**.

  Глава продолжает развивать информацию предыдущей главы в примерах, объясняющих, как использовать устройства, позволяющие Arduino определять тактильные, звуковые, тепловые, световые воздействия и положение предметов в пространстве.

- **Глава 7. Управление светодиодными устройствами вывода данных**.

  Работа со световыми устройствами. Примеры по включению одного или нескольких светодиодов и управление их цветом и яркостью. Объясняется управление светодиодными линейными индикаторами и цифровыми светодиодными дисплеями. Дается также общее представление о цифровом и аналоговом выводе данных.

- **Глава 8. Управление электродвигателями**.

  Придание объектам мобильности путем управления электродвигателями с помощью Arduino. Рассмотрение охватывает широкий диапазон разных типов электродвигателей: соленоидов, сервомашинок, двигателей постоянного тока и шаговых двигателей.

- **Глава 9. Работа со звуком**.

  Генерирование звука с помощью Arduino через выходные устройства типа динамиков. Воспроизведение простых тонов и мелодий, а также проигрывание файлов WAV и MIDI.

- **Глава 10. Дистанционное управление внешними устройствами**.

  Методы взаимодействия с практически любым устройством, оснащенным контроллером дистанционного управления какого-либо типа, включая телевизоры, аудиооборудование, фотокамеры, гаражные двери, бытовые устройства и игрушки. Примеры главы основаны на методах подключения Arduino к устройствам и модулям из предыдущих глав.

- **Глава 11. Работа с дисплеями**.

  Взаимодействие с текстовыми и графическими жидкокристаллическими и OLED-дисплеями. В частности, способы подключения этих устройств для вывода текста, прокрутки текста, а также выделения отдельных слов и создания специальных символов.

- **Глава 12. Работа с временем и датами.**

  Встроенные временны́е функции Arduino, а также многие дополнительные методы для обработки временны́х задержек, измерения периодов времени и реального времени и дат.

- **Глава 13. Протоколы связи I^2C и SPI.**

  Стандарты связи по протоколам I^2C (Inter-Integrated Circuit — межсхемный интерфейс интегральных схем) и SPI (Serial Peripheral Interface — последовательный синхронный периферийный интерфейс) предоставляют простые способы передачи данных с датчиков на плату Arduino. Рассматривается использование шин I^2C и SPI и, в частности, соединение двух или более плат Arduino с помощью шины I^2C.

- **Глава 14. Простая беспроводная связь.**

  Организация беспроводной связи устройств с помощью модулей XBee и Bluetooth. Представлены примеры их использования: от простой замены беспроводным подключением связи по последовательному порту до ячеистой сети для подключения множества датчиков к нескольким платам Arduino.

- **Глава 15. Сети Wi-Fi и Ethernet.**

  Способы использования платформы Arduino для работы в Интернете. Приводятся примеры, демонстрирующие создание и использование веб-клиентов и веб-серверов, а также применение наиболее распространенных протоколов связи Интернета на платформе Arduino, в том числе подключение платы Arduino к устройствам Интернета вещей.

- **Глава 16. Использование, модифицирование и создание библиотек.**

  Стандартным способом расширения функциональности среды Arduino является использование библиотек программ Arduino. В этой главе рассказывается, как применять и модифицировать готовые библиотеки, а также объясняется, как создавать свои собственные библиотеки.

- **Глава 17. Продвинутые методы программирования и работа с памятью.**

  Рассматриваются продвинутые методы программирования, а также темы, более технически углубленные, чем обсуждаемые при реализации предыдущих примеров, и затрагивающие вопросы, которые обычно скрыты под оболочкой Arduino. Эти методы можно использовать для повышения эффективности скетчей, улучшив их производительность и уменьшив объем кода.

- **Глава 18. Работа с аппаратными средствами микроконтроллера платы.**

  Получение доступа к аппаратным функциям микроконтроллера, которые не полностью доступны в среде программирования Arduino. В частности, рассматривается низкоуровневое использование аппаратных регистров ввода/вывода, таймеров и прерываний.

- **Приложение 1. Электронные компоненты.**

  Обзор электронных компонентов, используемых в примерах книги.

- **Приложение 2. Работа с принципиальными схемами и справочными листками.**

  Основные принципы работы с принципиальными схемами и справочными листками (datasheets).

- **Приложение 3. Сборка схем.**

  Краткое введение в работу с макетными платами, подключение и использование внешних источников питания и батарей, а также развязывающих конденсаторов.

- **Приложение 4. Советы по диагностированию программных неполадок.**

  Советы по поиску и устранению проблем с компиляцией и исполнением программ.

- **Приложение 5. Советы по диагностированию аппаратных неполадок.**

  Подходы к обнаружению и устранению аппаратных неисправностей.

- **Приложение 6. Цифровые и аналоговые контакты.**

  Таблицы назначения выводов (контактов) стандартных плат Arduino.

- **Приложение 7. Стандартный и расширенный коды ASCII.**

  Таблицы стандартного и расширенного кодов ASCII.

Что не вошло в эту книгу?

В рамках этой книги не представляется возможным подробно изложить теоретический и практический материал по электронике, хотя и предоставляются некоторые пояснения по сборке используемых в примерах схем. Более подробную теоретическую информацию в этой области можно легко найти в Интернете или в таких, например, книгах[2]:

- Чарльз Платт. Электроника для начинающих, 2-е изд. Издательство «БХВ-Петербург»[3].
- Getting Started in Electronics (Электроника для начинающих), автор Forrest M. Mims, III, издательство Master Publishing.
- Physical Computing (Физические вычисления), автор Tom Igoe, издательство Cengage.
- Саймон Монк, Пауль Шерц. Электроника. Теория и практика. Издательство «БХВ-Петербург»[4].
- Пауль Хоровиц, Уинфилд Хилл. Искусство схемотехники. Репринт оригинального издания издательства «Мир», 1986 г.[5]

---

[2] Если существует русское издание книги, упомянутой здесь авторами, в списке приводится ссылка именно на него.

[3] См. **https://bhv.ru/product/elektronika-dlya-nachinayushhih-2-e-izd/**.

[4] См. **https://bhv.ru/product/elektronika-teoriya-i-praktika/**.

В этой книге показано, как разрабатывать код для выполнения конкретных задач, но она не является учебником по программированию для начинающих на языках C или C++ (на которых основана среда разработки Arduino). Хотя в ней дается краткое объяснение использованных приемов программирования, осветить все детали в доступных рамках не представляется возможным. Для более глубокого изучения языков C/C++ можно порекомендовать одну из следующих книг:

- Дэвид Гриффитс, Дон Гриффитс. Изучаем программирование на C. Издательство «Эксмо»[6].
- Al Kelley и Ira Pohl. A Book on C (Учебник по C). Издательство Addison-Wesley.
- Брайан Керниган, Деннис Ритчи. Язык программирования C. Издательство «Вильямс»[7]. Популярный, хотя не совсем для начинающих, учебник, по которому многие люди научились программированию на языке C.
- Peter van der Linden. Expert C Programming: Deep C Secrets (Программирования на языке C для экспертов. Глубокие секреты языка C). Издательство Prentice Hall. В этом продвинутом, хотя и несколько устаревшем учебнике материал излагается занимательно и доходчиво, одновременно предоставляя информацию о том, почему язык C такой, каков он есть.

Запись кода программ

Код программ примеров в этой книге записан таким образом, чтобы ясно и четко иллюстрировать рассматриваемую в каждом примере тему. При этом в коде — особенно в начальных главах книги — не используются некоторые распространенные сокращения. Опытные программисты на языке C часто применяют мощные, но сжатые выражения, которые повышают эффективность кода, но часто трудно воспринимаются начинающими программистами. Так, в начальных главах книги инкрементирование переменных выполняется с использованием явных выражений, которые легко понимаются даже непрограммистами:

```
result = result + 1; // Инкрементируем счетчик
```

хотя применяемые опытными программистами более краткие выражения дают такой же результат:

```
result++; // Инкрементация, использующая постинкрементный оператор
```

Но это не означает, что вы должны использовать полный оператор — вы вполне свободны применять ту форму оператора, которая вам более удобна. При этом следует иметь в виду, что краткая форма операторов не предоставляет никаких особых преимуществ в виде лучшей производительности или меньшего объема компилированного кода.

---

[5] См. **https://www.bookvoed.ru/book?id=7382210**.

[6] См. **https://www.bookvoed.ru/book?id=648607**.

[7] **См. https://www.ozon.ru/context/detail/id/2480925/**.

Однако некоторые операторы употребляются настолько широко, что повсеместно используется их краткая версия. Например, краткая версия оператора цикла:

```
for(int i=0; i < 4; i++)
```

эквивалентна следующим двум строкам кода полной версии:

```
int i;
for(i=0; i < 4; i = i+1)
```

Более подробная информация по этим и другим операторам, используемым в этой книге, содержится в *главе 2*.

В число хороших практик программирования входит проверка на действительность используемых в вычислениях значений. Но чтобы сконцентрировать вас на теме примеров, их код содержит минимальное количество таких проверок.

Примечания к третьему изданию книги

Материал этого издания книги приведен и протестирован на соответствие версии 1.8.*x* среды разработки Arduino. Код примеров для этого издания также был обновлен должным образом. Загрузить код всех скетчей Arduino можно по адресу: **https://github.com/bjepson/Arduino-Cookbook-3ed-INO**, а скетчей Processing — по адресу: **https://github.com/bjepson/Arduino-Cookbook-3ed-PDE**.

> **Электронный архив**
>
> Электронный архив со скетчами для этой книги можно также загрузить с FTP-сервера издательства «БХВ» по ссылке **ftp://ftp.bhv.ru/9785977566872.zip** или со страницы книги на сайте **https://bhv.ru/**.

Веб-сайт английского издания этой книги (**https://oreil.ly/Arduino_Cookbook_3**) содержит ссылку на страницу списка обнаруженных в ней ошибок. С помощью этой страницы читатели могут сообщить нам об обнаруженных ими опечатках, ошибках и прочих проблемах с книгой. Указанные пользователями ошибки отображаются на вкладке **Unconfirmed** немедленно, а после проверки и подтверждения автором или редактором отображаются на вкладке **Confirmed**. Издательство O'Reilly может исправить обнаруженные ошибки в будущих изданиях книги и в обучающей платформе O'Reilly, таким образом весьма быстро улучшив качество обучающего материала.

Если у вас возникнут проблемы с работой приведенных в книге примеров, обратитесь к *приложению 4*, в котором рассматривается процесс диагностирования проблем в работе программ. Вы также можете обратиться за помощью к участникам форума Arduino (**https://forum.arduino.cc**), разместив на нем свой вопрос по наблюдаемой проблеме.

Если вам понравилась — или не понравилась — книга, дайте об этом знать другим людям в любом случае. Популярный способ поделиться своим мнением о книге — выложить его на странице книги веб-сайта Amazon.com. Отзывы о книге также можно оставить на онлайновой обучающей платформе O'Reilly.

Замечания и отзывы о русском издании этой книги читатели могут оставить на ее странице на сайте издательства «БХВ» по адресу: **https://bhv.ru/**.

Со времени выхода второго издания этой книги в мире микроконтроллеров в целом и в платформе Arduino в частности произошло много изменений: появилось большое количество новых плат, значительно возросли их вычислительные мощности, увеличился объем памяти, улучшились средства связи, уменьшились размеры плат и т. д. Хотя объем этой книги возрастает с каждым изданием, в ней все равно невозможно подробно рассмотреть все темы, которые могут интересовать всех читателей. Поэтому в этом издании мы сконцентрировались на обеспечении актуальности ее содержимого и обзоре широких возможностей, предоставленных сообществом Arduino со времени выхода предыдущего издания, чтобы помочь вам начать работать с этой замечательной технологией.

Обратите внимание: если вы используете более ранние версии платформы Arduino, чем рассматриваемые в этой книге, для них доступен код из первого и второго изданий этой книги. Загрузить этот код можно по адресам: **http://examples.oreilly.com/9780596802486** и **http://examples.oreilly.com/0636920022244** соответственно.

Стили типографского оформления

В книге применяется следующее типографское оформление:

- *Курсив*

  Обозначает новые термины.

- **Полужирный шрифт**

  Обозначает интернет-адреса (URL) и адреса электронной почты.

- Специальный шрифт Arial

  Обозначает названия и расширения файлов.

- `Моноширинный шрифт (с символами одинаковой ширины)`

  Используется для листингов программ, а также в тексте для обозначения элементов программ — таких как названия переменных или функций, баз данных, типов данных, переменных среды, операторов и ключевых слов.

- **`Полужирный моноширинный шрифт`**

  Используется для обозначения команд и прочих данных, которые вводятся пользователем.

- *`Курсивный моноширинный шрифт`*

  Обозначает текст, который заменяется пользователем своими значениями или значениями, определяемыми контекстом.

Такой значок обозначает подсказку или совет.

Такой значок обозначает общее примечание.

Такой значок обозначает предупреждение или необходимость обратить особое внимание.

Использование примеров кода

В случае возникновения технических вопросов касательно примеров кода или проблем с их использованием, можно задать соответствующий вопрос, отправив его по адресу электронной почты: **bookquestions@oreilly.com**.

Эта книга предназначена помочь вам реализовать свои проекты. Содержащийся в ней код примеров можно использовать в своих программах и документации без необходимости спрашивать нашего разрешения, если только речь не идет о значительном объеме кода. Например, использование нескольких фрагментов кода из этой книги в своей программе разрешения не требует, но для продажи или раздачи примеров из книг издательства O'Reilly такое разрешение требуется. При ответе на вопрос ссылка на эту книгу или цитирование кода примера разрешения не требует, но включение значительного объема кода из книги в документацию своего продукта требует разрешения.

Мы благодарны за ссылку на источник цитируемого текста, но обычно не требуем никакой ссылки. Формат такой ссылки обычно включает название книги, имя автора, издателя и ISBN книги. Например: «Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects. 3rd edition, авторы Майкл Марголис (Michael Margolis), Брайан Джепсон (Brian Jepson) и Николас Роберт Уэлдин (Nicholas Robert Weldin), издательство O'Reilly. Авторские права с 2020 года принадлежат Майклу Марголису, Николасу Роберту Уэлдину и Брайану Джепсону, ISBN 978-1-491-90352-0».

Если вы считаете, что использование вами кода примеров выходит за рамки добросовестного или приведенного здесь разрешения, можно уточнить этот вопрос, обратившись к нам по адресу электронной почты: **permissions@oreilly.com**.

Онлайновая программа обучения O'Reilly

Свыше 40 лет компания O'Reilly Media предоставляет обучение в области технологий и торогово-промышленной деятельности, а также знания и аналитическую информацию, помогающие компаниям добиваться успеха в своей деятельности.

Наша уникальная сеть экспертов и новаторов делится своими знаниями и опытом посредством книг, статей и нашей онлайновой обучающей платформы. Эта платформа открывает доступ по требованию к живым обучающим курсам, методам углубленного обучения, интерактивным средам программирования и огромному собранию текстов и видеоматериалов, предоставляемых издательством O'Reily и

свыше двухсот других издательств. Дополнительную информацию по этому вопросу можно найти на веб-сайте компании по адресу: **http://oreilly.com**.

Контакты

Если у вас есть какие-либо замечания или вопросы по этой книге, вы можете задать их издателю по следующему адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международный или местный) 707-829-0104 (факс)

Комментарии или технические вопросы по этой книге можно отправлять электронной почтой по адресу: **bookquestions@oreilly.com**.

Новости и дополнительную информацию о наших книгах и обучающих курсах можно найти на нашем веб-сайте (**http://www.oreilly.com**).

Мы в Facebook: **http://facebook.com/oreilly**.

В Twitter: **http://twitter.com/oreillymedia**.

И на YouTube: **http://www.youtube.com/oreillymedia**.

Замечания и отзывы о русском издании этой книги читатели могут оставить на ее странице на сайте издательства «БХВ» по адресу: **https://bhv.ru/**.

Благодарности от Майкла Марголиса за второе издание

Вклад Ника Уэлдина (Nick Weldin) в создание этой книги неоценим. Когда Ник присоединился к нашей команде, книга была готова на 90 процентов, и без его мастерства и энтузиазма она так бы готовой на 90 процентов и осталась. Его практический опыт управления семинарами по Arduino для пользователей всех уровней позволил нам сделать изложенные в книге советы практическими для нашего широкого диапазона читателей. Спасибо тебе, Ник, за твои знания, дружелюбный характер и дух коллективизма.

Редактор издательства O'Reilly Саймон Ст. Лоран (Simon St. Laurent) был первым, кто выразил заинтересованность в этой книге. И в конечном итоге он оказался тем, кто смог соединить все разрозненные части этого проекта в единое целое. Его поддержка и ободрение помогали нам сохранять нашу устремленность и вдохновение в процессе просмотра и анализа груды материала, необходимого, чтобы представить предмет книги должным образом.

Брайан Джепсон (Brian Jepson) помог мне приступить к написанию этой книги. Его громадные знания во всех областях платформы Arduino, его стремление обсуждать технологические вопросы на простом понимаемом языке и опыт в этой области были самого высшего класса. Он стал идеальной направляющей силой в наших уси-

лиях по приданию книге ее формы и предоставления читателям быстрого доступа к технологии. Мы также должны выразить Брайану нашу благодарность за новую информацию по радиомодулям XBee в *главе 14*.

Брайан Джепсон и Шон Уоллес (Shawn Wallace) были техническими редакторами второго издания книги и дали замечательные советы, как улучшить точность и ясность ее содержимого.

Одри Дойл (Audrey Doyle) неустанно трудилась над искоренением опечаток и грамматических ошибок в исходной рукописи и упрощением некоторых слишком запутанных выражений.

Филип Линдси (Philip Lindsay) сотрудничал с нами по подготовке содержимого *главы 15* первого издания книги. Ведущий разработчик многих улучшений в примерах по Ethernet первого издания книги Эдриен МакЮвен (Adrian McEwen) дал нам ценные советы по части отражения таких улучшений в этой главе.

Микал Харт (Mikal Hart) написал примеры по системе глобального позиционирования GPS и по программным средствам последовательной связи. Было вполне естественным пригласить Микала для решения этой задачи, и не только потому, что он разрабатывал соответствующие библиотеки, но также из-за его выдающихся коммуникативных способностей, большого энтузиазма по отношению к Arduino и уникальной направленности на совместную работу.

Платформа Arduino стала возможной благодаря креативному ядру команды ее разработчиков в составе: Массимо Банци (Massimo Banzi), Дэвида Квартиелеса (David Cuartielles), Тома Иго (Tom Igoe), Джанлуки Мартино (Gianluca Martino) и Дэвида Меллиса (David Mellis). От имени пользователей Arduino я хочу выразить нашу благодарность за их усилия, направленные на то, чтобы сделать эту захватывающую технологию простой, и за их великодушие в предоставлении свободного доступа к ней.

Особую благодарность хочу выразить Александре Дешампс-Сонсино (Alexandra Deschamps-Sonsino), чьи лондонские симпозиумы Tinker предоставили важную информацию для понимания требований пользователей. Кроме того, хочу поблагодарить Питера Найта (Peter Knight), который тоже предоставил разнообразные остроумные решения на основе Arduino, а также идеи для нескольких примеров в этой книге.

От имени всех загрузивших библиотеки Arduino, разработанные пользователями, хотел бы поблагодарить их разработчиков, которые великодушно поделились своими знаниями.

Наличие большого выбора аппаратного обеспечения является одним из важных факторов, которые делают платформу Arduino такой увлекательной. За это я хочу поблагодарить наших поставщиков, которые предоставили нам широкий диапазон великолепных устройств и всю необходимую техническую поддержку. Следующие поставщики оказали помощь в предоставлении аппаратных средств, использованных в проектах этой книги: SparkFun, Maker Shed, Gravitech и NKC Electronics. Помощь с предоставлением аппаратного обеспечения также предоставили компании

Modern Device, Liquidware, Adafruit, MakerBot Industries, Mindkits, Oomlout и SK Pang.

Ник хочет поблагодарить всех, кто занимался симпозиумами Tinker в Лондоне, в особенности таких людей, как Александр, Питер, Брок Крафт (Brock Craft), Даниэль Солтис (Daniel Soltis), а также всех, кто в течение многих лет помогал проводить симпозиумы.

Наконец, Ник выражает благодарность своей семье, Джини, Эмили и Финну, которые согласились позволить ему работать над этой книгой в течение летнего отпуска и, конечно же, намного дольше после него, чем они поначалу думали. Также он благодарит своих родителей Фрэнка и Еву, за то, что они взрастили его, чтобы он мог разбирать вещи на части.

Наконец, я выражаю свою благодарность следующим людям:

Джошуа Ноблу (Joshua Noble) — за то, что он представил меня издательству O'Reilly. Его книгу «Programming Interactivity» (Интерактивность программирования) настоятельно рекомендуется прочитать всем, кто заинтересован в расширении своих знаний в области интерактивных вычислений.

Роберту Лейси-Томпсону (Robert Lacy-Tompson) — за его советы на ранней стадии работы над первым изданием книги.

Марку Марголису — за его поддержку и помощь при обсуждении концепта книги и в процессе ее создания.

Я благодарю своих родителей за то, что они помогли мне понять, что творческое искусство и технология не являются строго отдельными сущностями и при их совместном использовании способны дать экстраординарные результаты.

Наконец, работу над этой книгой было бы невозможно начать и завершить без поддержки моей жены, Барбары Фаден (Barbara Faden). Я приношу ей искреннюю благодарность за поддержку моей мотивации и за ее внимательную вычитку рукописи и внесение своего вклада в нее.

Благодарности от Брайана Джепсона за третье издание

Сердечное спасибо Майклу Марголису, ведущему автору этой книги, и Джефу Блайлу (Jeff Bleiel), нашему редактору третьего издания. Они доверили мне встать во главе работы, чтобы представить вам это издание книги. Я ценю их доверие ко мне и уверенность в моих способностях и надеюсь, что они так же, как и я, довольны результатом. Я хочу поблагодарить свою жену, Джоан, за ее терпение и поддержку. Работа над книгой, особенно такой, для которой нужно собрать и протестировать десятки проектов, влияла на всех моих друзей и близких, и я высоко ценю понимание и поддержку, оказанную мне моими друзьями и членами моей семьи. Большое спасибо Крису Меринголо (Chris Meringolo) и Дону Колеману (Don Coleman) за техническую экспертизу, которая помогла мне организовать все работы таким образом, чтобы изложить материал в этой книге наиболее точно.

ГЛАВА 1

Начало работы

1.0. Введение

Платформа Arduino была разработана для того, чтобы начинающим пользователям, не обладающим опытом в области программирования или электроники, было легко работать с ней. С помощью платформы Arduino можно создавать устройства, которые будут реагировать на свет, звук, прикосновение или движение, и управлять ими. В частности, ее можно использовать для создания широкого диапазона устройств, включая музыкальные инструменты, роботов, световые арт-объекты, игры, интерактивную мебель и даже интерактивную одежду.

Платформа Arduino применяется во многих обучающих программах во всем мире, в особенности дизайнерами и творческими работниками, которые хотят иметь возможность быстро и легко создавать прототипы своих разработок, не требующие от них глубокого понимания технических подробностей их реализации. Поскольку платформа Arduino предназначена для использования людьми, не обладающими большим техническим опытом, ее программная часть содержит большой объем кода примеров, демонстрирующих использование различных возможностей платы Arduino.

Хотя с платформой Arduino легко работать, уровень сложности аппаратной части, лежащей в ее основе, тот же самый, что и у аппаратного обеспечения, используемого для создания промышленных встроенных устройств. Платформа Arduino привлекательна и для пользователей, которые уже работают с микроконтроллерами, — из-за предоставляемых ею возможностей быстрой разработки, позволяющих намного ускорить процесс воплощения идеи в рабочий прототип изделия.

Платформа Arduino славится своей аппаратной частью, но для программирования аппаратной составляющей требуются программные средства. Поэтому мы и называем оба эти компонента — и аппаратную, и программную составляющие платформы — общим названием Arduino. Оригинальные и совместимые платы Arduino (аппаратная составляющая) не стоят больших денег, а при желании плату Arduino — благодаря открытости аппаратной составляющей — можно собрать и самому. Программное обеспечение для Arduino вообще распространяется бесплатно, имеет открытый код и совместимо с основными операционными системами. Комбинация всех этих свойств платформы позволяет использовать ее для создания проектов, которые могут взаимодействовать с окружающим миром и управлять объектами в нем.

Кроме этого, существует активное и всегда готовое прийти на помощь сообщество Arduino, которое доступно во всем мире через форумы Arduino (**https://forum.arduino.cc**), учебные пособия (**https://oreil.ly/eptlu**) и центр проектов Project Hub (**https://oreil.ly/1aGpz**). На этих веб-сайтах можно найти обучающие ресурсы, примеры разработки проектов и решения задач, которые могут помочь вам в реализации своих собственных проектов.

Скетчи и программное обеспечение Arduino

Прикладные программы, которые на жаргоне Arduino называются *скетчами*, создаются на компьютере в интегрированной среде разработки (IDE) Arduino. Среда IDE позволяет создавать и редактировать код программ, а затем преобразовывать этот код в машинные инструкции для исполнения на плате Arduino. Среда IDE также используется для передачи этих машинных инструкций в виде компилированного кода на плату Arduino. Этот процесс называется *загрузкой скетча*.

Может быть, вы привыкли, что исходный код программы называется «программа» или просто «код». В сообществе Arduino исходный код, содержащий компьютерные инструкции для управления функциональностями Arduino, называется *скетчем*. Слово *скетч* будет использоваться в этой книге для обозначения исходного кода прикладных программ для Arduino.

Примеры из этой главы помогут вам приступить к работе, поясняя, как установить и настроить среду разработки Arduino и как скомпилировать и запустить на исполнение скетч примера.

Скетч Blink для мигания светодиодом, который установлен по умолчанию на большинстве оригинальных и совместимых плат Arduino, используется в этой главе в качестве эталонного примера. Но последний пример этой главы, кроме мигания светодиодом, обладает бóльшими способностями. В частности, с помощью дополнительных аппаратных средств, подключенных к плате Arduino, он может воспроизводить звуки и принимать входные данные.

Структура скетча Arduino рассматривается в *главе 2*, где и дается введение в программирование.

Если вы уже обладаете базовыми знаниями по Arduino, вы вполне можете пропустить этот материал и перейти к материалу, который вам еще не знаком. Но если это ваше первое знакомство с Arduino, то терпеливое изучение этих начальных примеров поможет вам разрабатывать свои скетчи с большей уверенностью в дальнейшем.

Аппаратная часть платформы Arduino

Код написанной программы исполняется на плате Arduino. Эта плата может реагировать только на электрические сигналы и управлять только ими, поэтому для придания ей возможности взаимодействия с внешним миром к ней подключаются различные дополнительные аппаратные устройства. Такими устройствами могут быть датчики, которые преобразовывают различные внешние воздействия в электриче-

ские сигналы, подаваемые затем на плату Arduino. А подключенные к плате исполнительные устройства преобразовывают генерируемые платой Arduino электрические сигналы в какой-либо тип физического действия. Примером датчиков могут служить переключатели, акселерометры, ультразвуковые дальномеры и т. п., а исполнительных устройств — светодиоды, динамики, электродвигатели, дисплеи и т. п.

На рынке предлагается большой выбор оригинальных плат Arduino и еще больший выбор совместимых плат, изготавливаемых как компаниями, так и отдельными членами сообщества Arduino. Установленные в платах Arduino микроконтроллеры используются, кроме самих плат, также в широком диапазоне других устройств — от 3D-принтеров до роботов. Некоторые из этих совместимых с Arduino плат и устройств также совместимы с другими средами программирования — такими как MicroPython или CircuitPython.

Самые популярные платы Arduino оснащены разъемом USB, который служит одновременно как для подачи питания на плату, так и для взаимодействия платы с компьютером, — в частности, для загрузки в плату скетчей. На рис. 1.1 показана базовая плата Arduino Uno, которая используется большинством начинающих пользователей. На плате установлен 8-разрядный микропроцессор ATmega328P, несущий на борту 2 Кбайт памяти SRAM (статическая память с произвольным доступом, служащая для хранения переменных программы) и 32 Кбайт флеш-памяти — для хранения скетчей. Работает этот процессор на частоте 16 МГц. Для обеспечения интерфейса USB на плате Arduino Uno установлен дополнительный микроконтроллер.

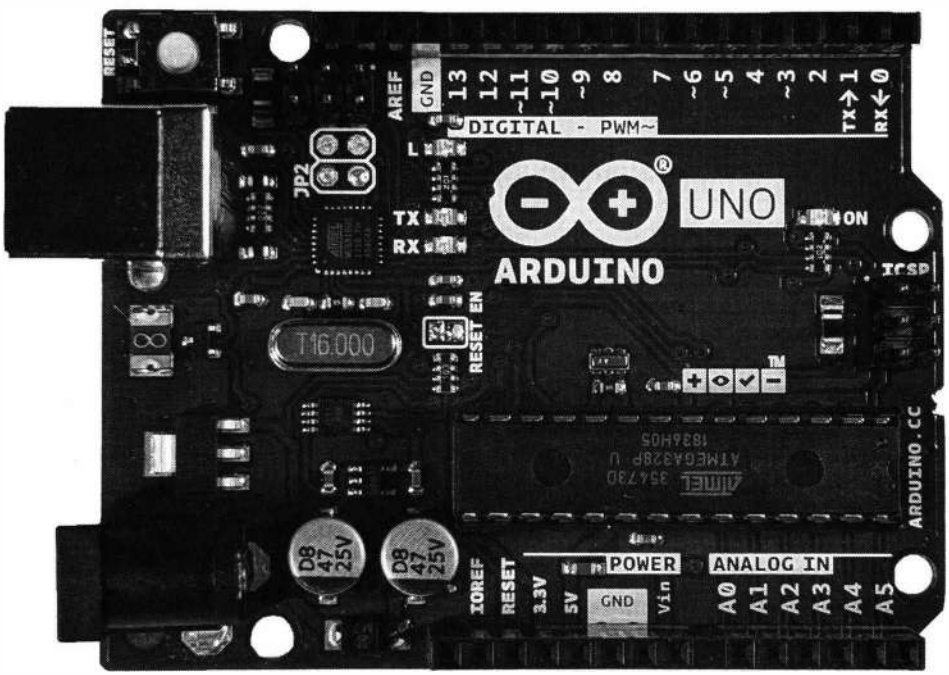


Рис. 1.1. Базовая плата Arduino Uno

Плата Arduino Leonardo имеет такой же *форм-фактор* (размер и организацию платы и ее контактов), что и плата Arduino Uno, но в ней установлен другой микроконтроллер — ATmega32U4. Этот микроконтроллер используется как для исполнения скетчей, так и для обеспечения интерфейса USB. Плата Arduino Leonardo чуть дешевле, чем плата Arduino Uno, но также поддерживает несколько интересных возможностей — таких как эмулирование различных USB-устройств, например клавиатуры и мыши. Совместимые с Arduino платы Teensy и Teensy++ компании PJRC (**http://www.pjrc.com/teensy**) также могут эмулировать устройства USB.

Плата Arduino Zero имеет похожее размещение контактов и даже более быстрый микроконтроллер. Но, в отличие от плат Uno и Leonardo, на контакты этой платы нельзя подавать сигналы напряжением выше чем 3,3 В. Плата Arduino Zero оснащена 32-разрядным микроконтроллером, который имеет 32 Кбайт памяти RAM и 256 Кбайт флеш-памяти и работает на частоте 48 МГц.

Плата Metro M0 Express компании Adafruit и плата RedBoard Turbo компании SparkFun имеют такой же форм-фактор, что и плата Arduino Zero, и также совместимы с разными средами разработки, включая среду Arduino IDE и CircuitPython.

Arduino и USB

Плата Arduino Uno оснащена дополнительным микроконтроллером для обеспечения интерфейса USB. Этот микроконтроллер в версии для поверхностного монтажа (ATmega16U2 или ATmega8U2 — в более старых версиях платы Uno) установлены на плате возле разъема USB. А плата Arduino Leonardo имеет только один микроконтроллер ATmega32U4, который отвечает как за исполнение кода скетчей, так и за интерфейс USB. Плату Leonardo можно перепрограммировать, чтобы она эмулировала другие устройства USB (см. *разд. 18.14*).

Более старые платы Arduino и некоторые совместимые с Arduino платы используют микросхему компании FTDI, которая реализует на аппаратном уровне подключение USB через последовательный порт компьютера. Некоторые дешевые клоны платы Arduino, которые предлагаются на площадках eBay или Amazon, могут использовать другую микросхему, например CH340, которая реализует такую же функцию. Для таких плат может потребоваться установить соответствующий драйвер.

Также существует класс совместимых с Arduino плат с поддержкой USB-интерфейса без использования аппаратных средств. Вместо этого в таких платах задействован программный подход, когда исполняющаяся на плате программа управляет контактами ввода/вывода для обмена сигналами USB. Такие платы, в число которых входит популярная оригинальная плата Trinket компании Adafruit, могут испытывать проблемы с работой при подключении к современным компьютерам, хотя с более старыми компьютерами могут работать должным образом. (Компания Adafruit выпустила плату Trinket M0, оснащенную аппаратной поддержкой USB, и которая, как бонус, намного быстрее, чем ее предшественница.)

Наконец, встречаются совместимые с Arduino платы, вообще не поддерживающие USB-подключение. Вместо этого они оснащены только контактами последовательного интерфейса, и для подключения к компьютеру через эти контакты требуют специального адаптера. Дополнительная информация по некоторым таким адаптерам приводится в *разд. «Аппаратные средства последовательной связи» главы 4*.

Если вы хотите иметь плату, пригодную для изучения возможностей Arduino, и на которой можно исполнить большинство скетчей из этой книги, плата Uno — прекрасный выбор для этих целей. Но если вам нужна плата, обладающая более высоким быстродействием, чем Uno, но при этом имеющая такой же форм-фактор, тогда обратите внимание на плату Zero или подобные ей платы — например, на плату

Metro M0 Express или RedBoard Turbo. Платы ряда MKR и Nano 33 также обладают отличным быстродействием, но имеют меньший форм-фактор, чем плата Uno.

Будьте осторожны с некоторыми платами, имеющими напряжение рабочего питания 3,3 В

Многие новые платы использует рабочее напряжение величиной 3,3 В, а не 5 В, как более старые платы, такие как Uno. Такие платы могут быть безвозвратно повреждены, если на какой-либо из их контактов ввода/вывода подать сигнал напряжением 5 В даже на малую долю секунды. Поэтому, если существует возможность подачи на контакты ввода/вывода такой платы сигнала напряжением выше чем 3,3 В, проверьте в документации на плату, способна ли она выдержать такое напряжение. В большинстве плат с рабочим напряжением 3,3 В на входной разъем рабочего напряжения подается напряжение величиной 5 В (например, через порт USB), но это напряжение понижается преобразователем напряжения до 3,3 В, прежде чем подаваться на электронные компоненты платы. В результате этого платы, чьи контакты ввода/вывода не допускают сигналов напряжением 5 В, весьма часто имеют контакт с выходным рабочим питанием 5 В.

Существуют платы Arduino и других форм-факторов, а это значит, что организация контактов на таких платах иная, чем у платы Uno, и они не совместимы с шилдами, предназначенными для работы с платой Uno. Плата MKR1010 — одна из таких плат Arduino с иным форм-фактором, намного меньшим, чем у платы Uno. Контакты ввода/вывода этой платы могут выдерживать сигналы напряжением только 3,3 В и в ней, так же как и в плате Zero, используется процессор ARM. Но плата MKR1010 оснащена встроенной поддержкой Wi-Fi и схемой для работы от литий-полимерной батареи и зарядки этой батареи. Хотя платы семейства MKR несовместимы с шилдами, предназначенными для платы Uno, для них предлагается свой выбор расширительных плат, называемых *носителями* (carriers).

Расширение возможностей Arduino с помощью шилдов

Функциональность плат Arduino можно расширить с помощью расширительных модулей, называемых *шилдами*. Эти модули устанавливаются поверх платы Arduino, при этом их штыревые разъемы вставляются в гнездовые разъемы платы Arduino. Для разных моделей плат Arduino и некоторых совместимых с Arduino плат могут предлагаться расширительные модули, похожие на шилды, но несовместимые с ними. Это объясняется использованием в этих платах иного форм-фактора, чем в самой популярной плате Arduino Uno. Например, плата Arduino MKR имеет меньшие физические размеры, чем плата Uno. Платы расширения для плат MKR также называются шилдами, хотя их форм-фактор несовместим с платой Uno. Компания Adafruit предлагает для своих плат громадный выбор плат расширения Featherwing разработчика линейки Feather, которые совместимы с программными средствами разработки Arduino. Однако платы расширения Featherwing несовместимы с платами Arduino иных форм-факторов — такими как платы Uno и MKR.

На рынке предлагаются платы размером с почтовую марку — например, плата Trinket M0 компании Adafruit. Платы же большего размера предоставляют более богатый выбор опций подключения и оснащаются более мощными процессорами — например, платы Arduino Mega и Arduino Due. Также имеются платы для специальных вариантов использования — например, носимая плата Arduino LilyPad, беспроводная плата Arduino Nano 33 IoT и встраиваемая плата Arduino Nano Every (для автономных приложений, которые часто питаются от батареи).

Многие сторонние производители также предлагают свои платы, совместимые с Arduino:

- несмонтированные платы — это наборы компонентов платы Arduino, включая саму печатную плату, которые пользователь должен собрать самостоятельно. Компания Modern Device (**https://oreil.ly/aBakM**) предлагает такие наборы с поддержкой интерфейса USB и без таковой, а компания Educato (**https://oreil.ly/bY5YZ**) — наборы, совместимые с шилдами;
- платы компании Adafruit Industries (**http://www.adafruit.com**) — компания Adafruit предлагает огромный выбор как оригинальных, так совместимых плат Arduino и вспомогательных модулей и компонентов;
- платы компании SparkFun (**https://oreil.ly/rr5Ry**) — компания SparkFun предлагает большой выбор модулей и компонентов как для плат Arduino, так и для совместимых плат;
- платы компании Seeed Studio (**http://www.seeedstudio.com**) — компания Seeed Studio предлагает как оригинальные, так и совместимые платы Arduino, а также множество модулей и компонентов к этим платам. У них также можно приобрести гибкую расширительную систему Grove (**https://oreil.ly/pSKC8**) для плат Arduino и других автономных плат, предлагающую варианты модульных разъемов для подключения датчиков и приводов;
- платы Teensy and Teensy++ (**http://www.pjrc.com/teensy**) — эти крошечные, но в высшей степени многоцелевые платы, предлагаются компанией PJRC.

Исчерпывающий список совместимых с Arduino плат можно найти в Википедии (**https://oreil.ly/uyFoP**). А на веб-сайте Arduino (**https://oreil.ly/yay5b**) приводится обзор плат Arduino.

1.1. Установка интегрированной среды разработки Arduino IDE

ЗАДАЧА

Установить на свой компьютер интегрированную среду разработки Arduino IDE.

РЕШЕНИЕ

Загрузите установочный пакет интегрированной среды разработки Arduino IDE со страницы загрузок веб-сайта Arduino (**http://arduino.cc/download**) для своей операционной системы — Windows, macOS или Linux. Далее приводятся инструкции для установки среды на каждую из этих платформ.

Установка среды Arduino IDE на Windows

Если вы работаете на компьютере под Windows 10, среду разработки Arduino IDE установить на него без прав администратора можно, используя магазин Microsoft

Store. Но для более ранних версий Windows запуск загруженного установщика среды возможен только с правами администратора. Однако вместо установочного пакета можно загрузить ZIP-архив среды и распаковать его в любую папку, для которой у вас есть права записи.

В результате распаковки вы получите папку с названием Arduino-<nn>, где <nn> означает номер версии загруженной среды разработки Arduino IDE. Эта папка содержит исполняемый файл Arduino.exe и прочие файлы и папки среды.

При первом запуске среды Arduino IDE может открыться диалоговое окно с сообщением, что брандмауэр Защитника Windows заблокировал некоторые возможности этого приложения, указывая на файл *javaw.exe* как на причину такого предупреждения. То, что предупреждение вызывается программой *javaw.exe*, а не программой *Arduino.exe*, объясняется тем, что среда разработки Arduino IDE основана на платформе Java. Приложение *javaw.exe* используется платами, поддерживающими возможность загрузки в них скетчей по локальной сети. Если вы планируете работать с такими платами, вам нужно будет в окне предупреждения разрешить этому приложению доступ к локальной сети.

При подключении платы Arduino к компьютеру система автоматически ассоциирует установленные в ней драйверы с платой (это может занять некоторое время). Если же этот процесс не завершится успехом, или если вы установили среду Arduino IDE из ZIP-архива, вам придется установить драйверы для своей платы вручную. Для этого посетите веб-страницу Getting Started with Arduino products (**https://oreil.ly/ELrle**), в правой панели этой страницы щелкните на ссылке со своей платой и установите драйверы, следуя предлагаемым инструкциям.

Если вы используете более старую плату (т. е. любую плату с драйверами FTDI) и ваш компьютер подключен к Интернету, выполнение поиска драйверов для платы можно предоставить Windows, и они установятся автоматически. Если подключение к Интернету отсутствует, или если используется операционная система Windows XP, место расположения драйверов надо будет указать вручную. Для этого с помощью окна навигации по файловой системе выберите папку drivers\FTDI USB Drivers в папке, в которую вы распаковали ZIP-архив. После установки драйвера платы откроется диалоговое окно **Обнаружено новое оборудование**, сообщая об обнаружении нового последовательного порта. Снова выполните описанный ранее процесс установки драйвера.

Возможно, вам придется выполнить процесс установки драйверов дважды, чтобы среда Arduino IDE могла взаимодействовать с платой.

Установка среды Arduino IDE на macOS

Для компьютеров под управлением macOS среда разработки Arduino IDE загружается в виде ZIP-архива. Если загруженный архив не распакуется автоматически, перейдите в папку, в которую он был загружен, и щелкните на нем двойным щелчком мыши, чтобы его распаковать. Переместите распакованную папку среды в какую-либо удобную папку — например, в папку Приложения, где ей самое место.

Запустите среду на исполнение, щелкнув двойным щелчком мыши на ее значке. Откроется начальный экран среды, после которого откроется главное окно программы.

Для современных плат Arduino — таких как Uno, установка дополнительных драйверов не требуется. При первом подключении платы к компьютеру может открыться диалоговое окно с сообщением, что обнаружен новый сетевой порт. Его можно закрыть, не предпринимая никаких действий. Для более старых плат может потребоваться установить драйверы FTDI, которые можно загрузить на веб-сайте этой компании (**https://oreil.ly/w7_YM**).

Установка среды Arduino IDE на Linux

Версии среды разработки Arduino IDE для Linux все чаще включаются разработчиками в пакет установочного дистрибутива этой операционной системы. Но обычно это не самые последние версии среды, поэтому лучше всего загрузить ее текущую версию с веб-сайта Arduino по адресу: **http://arduino.cc/download**. Среда Arduino IDE для Linux доступна как для 32- так и для 64-разрядных версий системы, предлагается также версия и для процессоров ARM, которую можно установить на одноплатные компьютеры Raspberry Pi и другие компьютеры с процессорами линейки ARM и операционной системой Linux. Большинство дистрибутивов используют стандартный предустановленный драйвер, а также обычно поддерживают интерфейс FTDI. Подробные инструкции по установке среды Arduino IDE на компьютеры под Linux вы найдете на веб-сайте Arduino по адресу: **https://www.arduino.cc/en/guide/linux**. Вкратце: вам надо будет запустить на исполнение сценарий установки, а также может потребоваться дать разрешение своей учетной записи на доступ к последовательному порту.

* * *

Для совместимых с Arduino плат сторонних производителей может потребоваться установить файлы поддержки, используя для этого Менеджер плат (см. *разд. 1.7*). При возникновении необходимости в процессе установки среды Arduino IDE выполнить какие-либо дополнительные шаги для конкретной платы, обратитесь за информацией к документации для этой платы.

Завершив установку среды Arduino IDE, запустите ее на исполнение, щелкнув на значке программы двойным щелчком, — в результате должно открыться окно начальной заставки (рис. 1.2).

За начальным экраном-заставкой следует главное окно программы, показанное на рис. 1.3. Для открытия главного окна может потребоваться некоторое время, чтобы загрузился требуемый код, поэтому запаситесь терпением. Значок для запуска среды Arduino IDE должен присутствовать в меню **Пуск | Все программы** Windows и в папке Приложения macOS, а также, возможно, на рабочем столе компьютера. На системах под Linux среда Arduino IDE запускается из оболочки терминала.

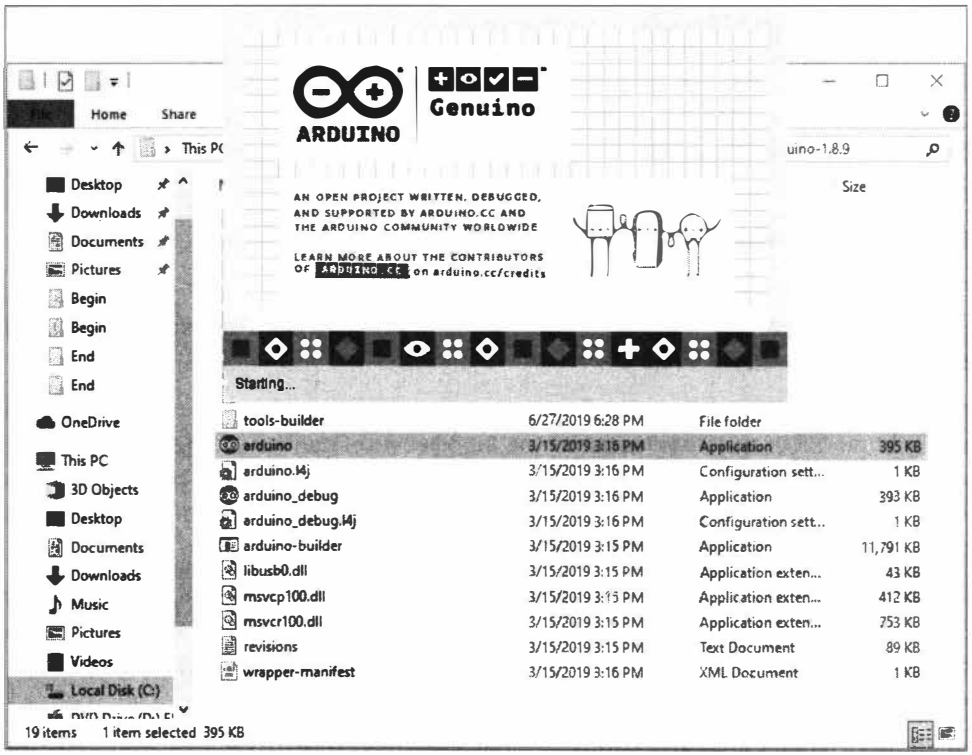


Рис. 1.2. Окно заставки при запуске среды Arduino IDE (Arduino 1.8.9 на Windows 10)

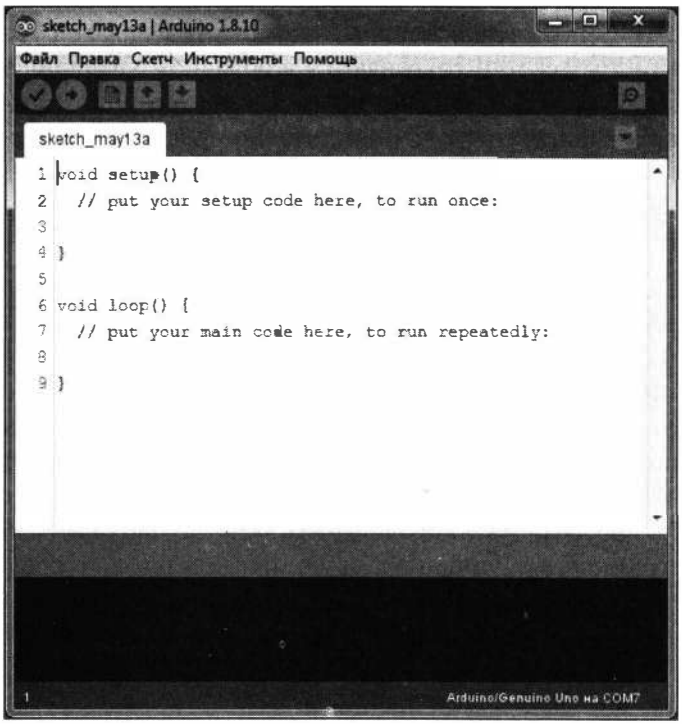


Рис. 1.3. Главное окно среды Arduino IDE (Arduino 1.8.9 на macOS)

Возможные проблемы установки

Если среда Arduino IDE не запускается, инструкции по диагностированию возможных причин этого можно найти в разделе устранения неполадок веб-сайта Arduino по адресу: **https://oreil.ly/sIIus**.

Дополнительная информация

На веб-сайте Arduino приводятся подробные инструкции по установке среды Arduino IDE для Windows (**https://oreil.ly/ufBhk**), macOS (**https://oreil.ly/GL4uk**) и Linux (**https://oreil.ly/TW6si**).

Кроме стандартной среды разработки Arduino IDE также может использоваться среда Arduino Pro, ориентированная на профессиональных пользователей. Но на момент подготовки книги эта среда была только в начальной степени готовности. Дополнительную информацию по среде Arduino Pro можно найти на веб-странице ее репозитория GitHub (**https://oreil.ly/tKGNk**).

Инструмент командной строки Arduino CLI предназначен для компилирования и загрузки скетчей. Его также можно использовать вместо Менеджера библиотек и Менеджера плат. Дополнительную информацию по этому инструменту вы найдете на веб-странице его репозитория GitHub (**https://oreil.ly/yaOph**).

Наконец, доступна также и онлайновая среда разработки Arduino Create. Чтобы работать с этой средой, необходимо создать на веб-сайте Arduino учетную запись и установить на свой компьютер программный подключаемый модуль, который позволяет веб-сайту взаимодействовать с подключенной к компьютеру платой для загрузки в нее кода. Разрабатываемые скетчи сохраняются в облачном хранилище, и доступ к ним можно предоставлять другим разработчикам. На момент подготовки этой книги среда Arduino Create представляет собой достаточно новый сервис, все еще находящийся в стадии развития. Впрочем, если вы хотите разрабатывать скетчи Arduino, не устанавливая среду разработки на свой компьютер, среда Arduino Create (**https://create.arduino.cc**) сможет вам в этом помочь.

Для пользователей Chromebook в среде Arduino Create предлагается приложение Chrome, доступ к которому требует подписки стоимостью $1 в месяц. Впрочем, это приложение предусматривает также и временный бесплатный доступ, чтобы можно было с ним ознакомиться. Альтернативу среде Arduino Create для компилирования и загрузки кода с устройств Chromebook предлагает и онлайновая среда разработки Codebender. Эта среда также поддерживает некоторые совместимые с Arduino платы сторонних производителей. Бесплатная версия среды имеет ограниченные возможности, которые снимаются при оформлении одного из типа подписок — для класса, школы или школьного округа. Дополнительную информацию о среде разработки Codebender можно найти на веб-сайте этого сервиса (**https://edu.codebender.cc**).

1.2. Подготовка платы Arduino к работе

ЗАДАЧА

Подать питание на плату Arduino и удостовериться, что она работает должным образом.

РЕШЕНИЕ

Подключите плату к порту USB компьютера и проверьте, загорелся ли на ней светодиодный индикатор питания (рис. 1.4). Большинство плат Arduino оснащены таким индикатором, который загорается при подаче питания на плату.

Кроме этого, на плате должен начать мигать другой светодиодный индикатор. Большинство плат Arduino при изготовлении программируются простым скетчем, который мигает этим светодиодом, осуществляя тем самым простую проверку работоспособности платы.

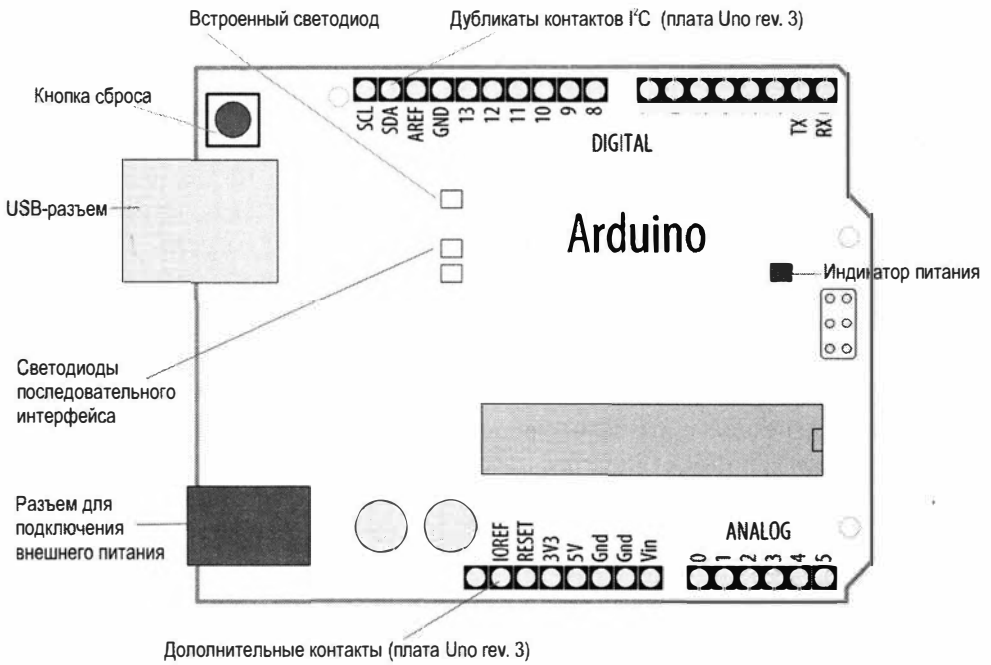


Рис. 1.4. Схема базовой платы Arduino Uno (Rev. 3)

На современных версиях платы Arduino Uno присутствуют несколько контактов, которых нет в старых версиях платы. Соответствующие контакты могут отсутствовать и в некоторых старых шилдах Arduino. К счастью, это обычно не сказывается на использовании старых шилдов, и большинство из них будут работать с новыми версиями платы так же, как и со старыми (но результаты могут быть разными для разных шилдов).

Дополнительный контакт IOREF в новых версиях платы служит для предоставления шилдам аналогового эталонного напряжения, чтобы входные аналоговые значения можно было соотнести с напряжением источника питания. А контакты SCL и SDA добавлены с целью обеспечить постоянное расположение контактов этих сигналов для устройств I^2C. Вследствие разных конфигураций микроконтроллеров на некоторых более старых версиях плат (например, Mega) контакты I^2C могли располагаться в разных местах и для некоторых шилдов нужно было прибегать к обходным маневрам — например, подключать контакты I^2C шилда к соответствующим контактам платы с помощью проволочных перемычек. Теперь же шилды, разработанные под новую организацию контактов, должны работать со всеми платами с соответствующими дополнительными контактами. В настоящее время дополнительный контакт рядом с контактом IOREF не используется, но может быть задействован для реализации в будущем какой-либо новой функциональности, не требующей снова изменять организацию контактов.

Обсуждение возможных проблем

Если при подключения платы к компьютеру ее индикатор питания не загорается, это, скорее всего, потому, что на плату не подается питание. Попробуйте подключить плату к другому разъему USB компьютера или с помощью другого кабеля USB.

Мигающий светодиод управляется исполняющимся на плате скетчем Blink, который загружается в плату при ее изготовлении. Мигание означает, что этот скетч исполняется должным образом, что, в свою очередь, свидетельствует о правильной работе платы. Если светодиод индикатора питания горит, но светодиод (обычно обозначенный буквой L) не мигает, причина этого может заключаться в том, что скетч Blink не был в плату загружен. Следуя инструкциям из *разд. 1.3*, загрузите в плату этот скетч, чтобы удостовериться в работоспособности платы. На платах иных, чем Uno, встроенный светодиод может отсутствовать — сверьтесь с документацией на свою плату, чтобы узнать, оснащена ли она таким светодиодом.

На платах Leonardo и платах класса Zero (Arduino Zero, Adafruit Metro M0, SparkFun RedBoard Turbo) гнездовые разъемы расположены так же, как и на плате Uno, позволяя подключение к ним стандартных шилдов. Но в прочих моментах эти платы значительно отличаются от платы Uno. Подобно плате Uno, плата Leonardo оснащена 8-разрядным микроконтроллером, но поскольку она не имеет отдельного микроконтроллера для обеспечения интерфейса USB, программы в нее можно загружать только в течение некоторого периода времени сразу же после сброса. Период времени, в течение которого плата Leonardo ожидает загрузки скетча, обозначается миганием встроенного светодиода платы. Плата Leonardo может работать с логическими сигналами напряжением 5 В.

В плате Zero используется 32-разрядный микропроцессор ARM, имеющий больший объем памяти для хранения и исполнения скетчей. Плата также оснащена встроенным цифроаналоговым преобразователем (ЦАП), выходной сигнал которого подается в виде варьирующегося напряжения на отдельный контакт ввода/вывода платы. С помощью ЦАП можно генерировать звуковые сигналы намного лучшего качества, чем посредством платы Uno. Плата Zero не поддерживает работу с логи-

ческими сигналами напряжением 5 В, так же как и подобные платы компании Adafruit (Metro M0 Express) и компании SparkFun (RedBoard Turbo).

Плата Arduino MKR1010 оснащена таким же микропроцессором, что и плата Zero, и также не поддерживает работу с логическими сигналами напряжением 5 В. При этом она имеет еще и меньший форм-фактор, т. е. несовместима с шилдами, предназначенными для плат форм-фактора Uno. Эта плата также поддерживает Wi-Fi, что позволяет подключать ее к Интернету через сеть Wi-Fi.

Все 32-разрядные платы имеют большее количество контактов, поддерживающих прерывания, чем большинство 8-разрядных плат. Прерывания полезны в приложениях, которые должны быстро определять изменения уровней сигнала (см. *разд 18.2*). Единственным исключением из этого является плата Arduino Uno WiFi Rev2, которая поддерживает прерывания на всех своих цифровых контактах ввода/вывода.

Дополнительная информация

На веб-сайте Arduino приводятся подробные инструкции по установке среды Arduino IDE для Windows (**https://oreil.ly/ufBhk**), macOS (**https://oreil.ly/GL4uk**) и Linux (**https://oreil.ly/TW6si**). Инструкции для конкретной платы можно найти на странице Arduino Guide (**https://oreil.ly/MHLdN**).

На веб-сайте Arduino (**https://oreil.ly/lYeR0**) также приводится руководство по диагностированию проблем.

1.3. Создание скетча в среде разработки Arduino IDE

ЗАДАЧА

Ознакомиться с процессом компилирования скетчей в среде Arduino IDE и разобраться с сообщениями об ошибках и статусными сообщениями.

РЕШЕНИЕ

Исходный код программы Arduino называется *скетчем*. Процесс преобразования исходного кода скетча в форму, которая может исполняться платой Arduino, называется *компиляцией*. Для создания скетчей, открытия и модифицирования скетчей, которые определяют поведение платы Arduino, используется среда разработки Arduino IDE. Все указанные действия можно выполнять с помощью кнопок, расположенных в верхней части главного окна среды Arduino IDE (рис. 1.5). Впрочем, вы можете также воспользоваться меню (рис. 1.6) или соответствующими клавишами быстрого вызова.

Для ввода и редактирования кода скетча предназначена область редактора, которая занимает бóльшую часть главного окна среды Arduino IDE. Редактор поддерживает распространенные клавиши быстрого вызова для редактирования текста:

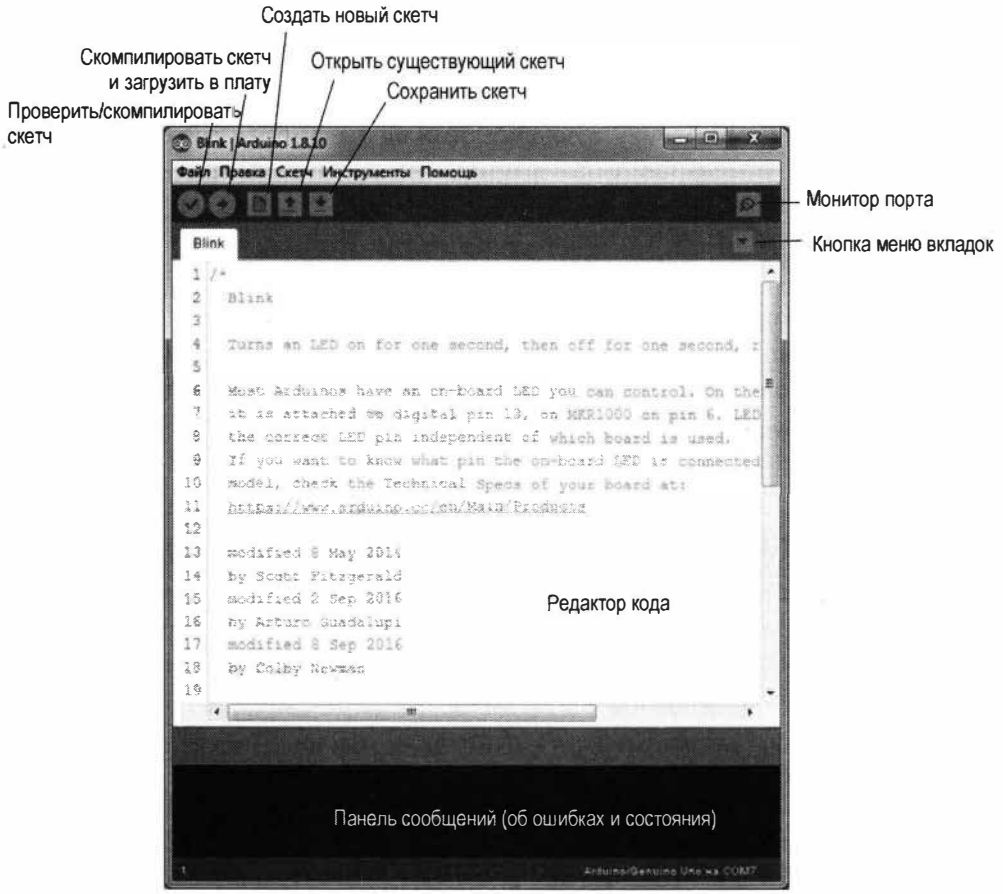


Рис. 1.5. Элементы панели инструментов и окна среды Arduino IDE

- <Ctrl>+<F> (<Command>+<F> на macOS) — для поиска текста;
- <Ctrl>+<Z> (<Command>+<Z> на macOS) — для отмены последней операции;
- <Ctrl>+<C> (<Command>+<C> на macOS) — для копирования выделенного текста в буфер обмена;
- <Ctrl>+<V> (<Command>+<V> на macOS) — для вставки скопированного текста.

На рис. 1.6 показано, как загрузить скетч Blink в плату Arduino. (Как уже было отмечено ранее, этот скетч загружается в современные платы Arduino при их изготовлении.)

Запустив среду Arduino IDE, выполните последовательность команд меню **Файл** | **Примеры** | **01.Basics** и выберите в списке скетч **Blink** (см. рис. 1.6) — откроется новое окно редактора, содержащее код для мигания встроенным светодиодом (см. рис. 1.5).

Но чтобы плата могла исполнить код скетча, его нужно преобразовать в инструкции, которые может понимать и исполнять главный микроконтроллер платы

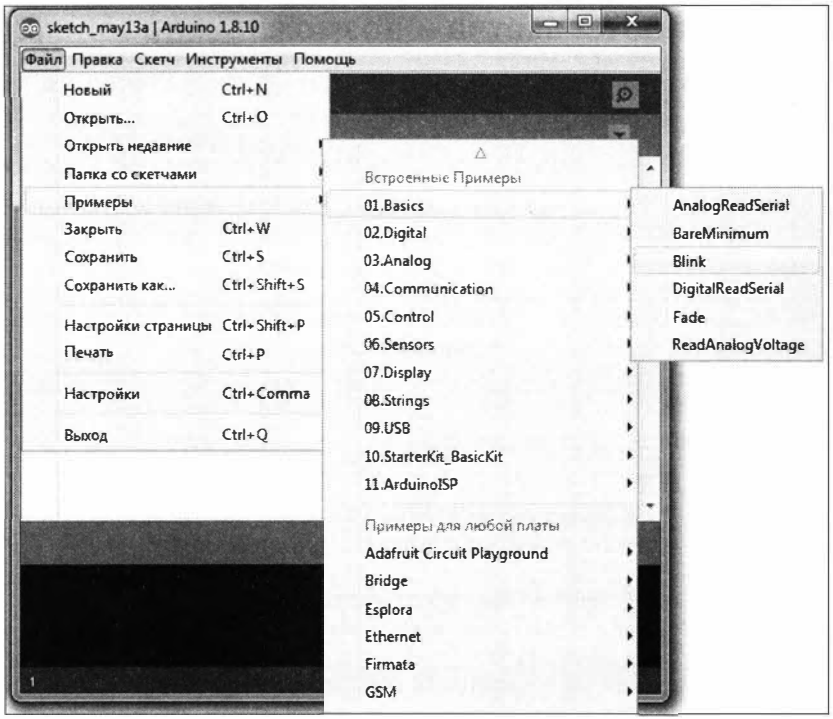


Рис. 1.6. Выбор скетча Blink для загрузки в плату Arduino с помощью меню среды Arduino IDE (на Windows 10)

Arduino. Этот процесс преобразования называется *компиляцией*. Процесс компиляции можно запустить, нажав кнопку **Проверить** (самая левая кнопка окна с галочкой в ней), выполнив последовательность команд меню **Скетч | Проверить/Компилировать** или нажав клавиши быстрого вызова <Ctrl>+<R> (<Command>+<R> на macOS).

В панели сообщений, расположенной ниже окна редактора кода, должно отобразиться сообщение **Компиляция скетча**, а справа от сообщения — индикатор выполнения. Компиляция должна занять пару секунд, после чего в верхней области панели сообщения будет выведено сообщение **Компиляция завершена**, а в самой панели сообщения (область с черным фоном) отобразится более подробное сообщение следующего вида:

> Скетч использует 924 байта (2%) памяти устройства. Всего доступно 32 256 байтов. Глобальные переменные используют 9 байтов (0%) динамической памяти, оставляя 2039 байтов для локальных переменных. Максимум: 2048 байтов.

В зависимости от используемой платы Arduino и версии среды разработки Arduino IDE сообщение может быть слегка иным. Но в любом случае в нем предоставляется информация о размере скетча в байтах и максимальном размере скетча, который можно загрузить в плату.

Обсуждение работы скетча и возможных проблем

Для компиляции скетча среда разработки использует несколько инструментов командной строки. Более подробная информация об этом приводится в *разд. 17.1*.

В сообщении, которое выводится по завершении компилирования скетча, сообщается, какой объем программной памяти платы требуется для хранения скомпилированного кода программы. Если размер скомпилированного скетча превышает объем памяти на плате, будет выведено сообщение следующего вида:

```
Скетч слишком большой; обратитесь за советами по его уменьшению по адресу:
http://www.arduino.cc/en/Guide/Troubleshooting#size.
```

В таком случае нужно или уменьшить размер скетча, чтобы он помещался в доступную память на плате, или же использовать другую плату с большим объемом программной флеш-памяти. Возможна и ситуация, когда сам скетч помещается в память, но объем глобальных переменных слишком большой для отведенной для них памяти. При этом выводится сообщение следующего вида:

```
Недостаточно памяти; обратитесь за советами по адресу:
http://www.arduino.cc/en/Guide/Troubleshooting#size.
```

Тогда нужно проверить код и уменьшить объем памяти, выделяемый для глобальных переменных, или же использовать плату с большим объемом статической памяти SRAM.

Чтобы не допустить случайного затирания кода примеров, среда Arduino IDE не позволяет сохранять изменения в скетчах примеров, встроенных в среду. Чтобы сохранить измененный скетч примера, ему нужно присвоить другое название, воспользовавшись для этого опцией меню **Файл | Сохранить как**. Собственные скетчи сохраняются с помощью команды меню **Сохранить** или одноименной кнопки (см. *разд. 1.5*).

При наличии в коде синтаксических ошибок компилятор выводит соответствующие сообщения в главной области панели сообщений. Эти сообщения помогают определить причину ошибки, вызвавшей сообщение. Более подробная информация с советами по диагностированию ошибок в скетчах приводится в *приложении 4*.

Кроме сообщений об ошибках, компилятор может выводить предупреждения, если он решит, что некоторые аспекты скетча способны вызвать проблемы. Эти предупреждения могут быть очень полезными, поскольку дают вам возможность принять меры для устранения подозрительных фрагментов кода, которые могут вызвать проблемы в дальнейшем. Пользователь может установить удобный для него уровень предупреждений в настройках, выполнив команду меню **Файл | Настройки** (Window или Linux) или **Arduino | Настройки** (macOS) и выбрав в выпадающем списке **Сообщения компилятора** необходимую опцию: **Ничего, По умолчанию, Подробнее, Все**. Несмотря на наличие в списке опции **По умолчанию**, в действительности по умолчанию устанавливается опция **Ничего**. Рекомендуется установить опцию **По умолчанию** или **Подробнее**.

Код, загруженный в плату Arduino, нельзя выгрузить обратно в компьютер. Поэтому позаботьтесь о том, чтобы сохранить исходный код разрабатываемого скетча на своем компьютере. Как уже упоминалось ранее, модифицированный скетч примера нельзя сохранить под его исходным названием — ему нужно присвоить другое название, воспользовавшись для этого опцией меню **Файл | Сохранить как**.

Дополнительная информация

В *разд. 1.5* приводится пример скетча. В *приложении 4* содержатся советы по диагностированию программных неполадок.

1.4. Загрузка и исполнение скетча Blink

ЗАДАЧА

Скомпилировать и загрузить скетч в плату Arduino и убедиться, что он работает.

РЕШЕНИЕ

Подключите плату Arduino к компьютеру через разъем USB, используя для этого кабель USB. Откройте в редакторе скетчей среды Arduino IDE скетч Blink, выполнив команду меню **Файл | Примеры | 01. Basics** и выбрав в списке опцию **Blink**.

Выполните команду меню **Инструменты | Плата** и выберите название подключенной платы. (Если это стандартная плата Uno, то она, наверное, будет одной из первых в разделе **Платы Arduino AVR** предложенного списка.)

Затем выполните команду меню **Инструменты | Порт**, чтобы открыть список доступных последовательных портов на своем компьютере. В зависимости от подключенных периферийных устройств, разные компьютеры покажут в этом списке разные комбинации портов.

На компьютерах под Windows порты обозначаются буквами COM и номером порта — например, COM1, COM2 и т. д. Если в списке отображен только один порт, выберите именно его. Если список содержит несколько портов, то порт платы будет, скорее всего, самым последним в списке.

На компьютерах под macOS названия портов для платы Uno имеют следующий формат:

```
/dev/cu.usbmodem-XXXXXXX(Arduino/Genuino Uno)
```

Для более старых плат формат будет следующим:

```
/dev/tty.usbserial-XXXXXXX
/dev/cu.usbserial-XXXXXXX
```

Для каждой платы значение *XXXXXXX* будет другим. Вы можете выбрать любой из этих двух вариантов.

На компьютерах под Linux названия портов для платы Uno могут иметь следующий формат:

```
/dev/ttyACMX(Arduino/Genuino Uno)
```

Для более старых плат формат может быть следующим:

```
/dev/ttyUSB-X
```

где X обозначает собственно номер порта. Обычно это будет значение 0, но если одновременно подключено несколько плат, то также вы увидите значения 1, 2 и т. д. Выберите порт, соответствующий вашей плате Arduino.

Если список портов содержит несколько позиций, и вы не знаете, какая из них относится к вашей плате Arduino, определить ее можно следующим образом. Откройте — не подключая плату Arduino к компьютеру — список портов (**Инструменты | Порт**) и запомните или запишите присутствующие в нем порты. Затем подключите плату Arduino к компьютеру, и вы увидите новую позицию, добавленную в этот список после подключения платы. Это и есть порт для подключенной платы Arduino. Также можно определить требуемый порт последовательным перебором всех портов, пытаясь загрузить скетч в каждый выбранный Когда на плате замигают светодиоды последовательного интерфейса (см. рис. 1.4), это будет означать, что выбран правильный порт и код загружается в плату.

Нажмите кнопку **Загрузка** (на рис. 1.5 это вторая кнопка слева на панели инструментов) или выполните команду меню **Скетч | Загрузка** (или нажмите клавиши быстрого доступа: <Ctrl>+<U> на Windows, <Control>+<U> на macOS).

Среда разработки Arduino IDE выполнит компиляцию кода, как было показано в *разд. 1.3*, а по завершении компиляции загрузит скомпилированный код в плату Arduino. В случае абсолютно новой платы Arduino с уже загруженным в нее скетчем Blink встроенный светодиод (см. рис. 1.4) перестанет мигать. В процессе загрузки кода в плату будут мигать два светодиода последовательного интерфейса (см. рис. 1.4), расположенные возле встроенного тестового светодиода. По завершении загрузки светодиоды последовательного интерфейса перестанут мигать, начнет исполняться загруженный код, и снова начнет мигать встроенный тестовый светодиод. Расположение только что упомянутых светодиодов будет разным для различных плат Arduino — таких как Leonardo, MKR и клонов сторонних производителей.

Обсуждение работы скетча и возможных проблем

Чтобы среда разработки Arduino IDE могла загрузить скомпилированный код

в плату, плата должна быть подключена к порту USB компьютера. Кроме этого, среде Arduino IDE необходимо указать используемую плату и порт подключения.

Если на плате исполнялся какой-либо скетч, после начала загрузки скомпилированного кода скетча исполнение старого скетча прекращается. (В частности, если исполнялся скетч Blink, встроенный тестовый светодиод прекращает мигать.) Загруженный новый скетч заменяет старый и начинает исполняться после завершения загрузки.

Некоторые старые платы и клоны Arduino не прерывают автоматически исполнение старого скетча, чтобы начать загружать новый. В таком случае, чтобы загрузить новый скетч, нужно нажать кнопку сброса на плате Arduino сразу же после завершения компиляции, когда в панели сообщений отобразится соответствующее

сообщение. Чтобы уловить правильный момент для нажатия кнопки сброса после завершения компиляции, может потребоваться несколько попыток.

При проблемах с загрузкой скетча в плату в панели сообщений отобразится соответствующее сообщение. Проблемы с загрузкой обычно вызываются такими причинами, как указание неправильной платы или порта в настройках среды IDE или же тем, что плата Arduino не подключена к компьютеру. Текущая указанная плата и последовательный порт для нее отображаются в строке состояния в правой нижней части главного окна среды Arduino IDE (см., например, рис. 1.5).

Дополнительная информация

Дополнительную информацию по диагностированию проблем с загрузкой и исполнением скетчей Arduino можно найти на странице **Arduino Troubleshooting** вебсайта Arduino (**https://oreil.ly/lYeR0**).

1.5. Создание и сохранение скетча

ЗАДАЧА

Создать скетч и сохранить его на своем компьютере.

РЕШЕНИЕ

Запустите среду Arduino IDE (см. *разд. 1.3*) и откройте новое окно редактора, выполнив команду меню **Файл | Создать**. Удалите из поля редактора шаблонный код или код предыдущего скетча и вставьте в него код, приведенный в листинге 1.1, — это практически тот же скетч Blink, только он мигает светодиодом вдвое медленнее.

Листинг 1.1. Пример скетча

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

void loop()
{
   digitalWrite(LED_BUILTIN, HIGH); // Включаем встроенный светодиод
   delay(2000);                     // Ждем две секунды
   digitalWrite(LED_BUILTIN, LOW);  // Выключаем встроенный светодиод
   delay(2000);                     // Ждем две секунды
}
```

Проверьте правильность кода, скомпилировав его без загрузки в плату. Для этого нажмите кнопку **Проверить** (левая кнопка в панели инструментов окна среды

Arduino IDE) или выполните команду меню **Скетч | Проверить/Компилировать** (см. *разд. 1.3*).

В случае успешной компиляции загрузите скомпилированный код в плату Arduino, нажав кнопку **Загрузить** или выполнив команду меню **Скетч | Загрузка**. После успешной загрузки скетча, так же как и в случае со скетчем Blink, должен начать мигать встроенный тестовый светодиод, но только вдвое медленнее — один раз в две секунды.

На локальный компьютер скетч сохраняется выполнением стандартной процедуры сохранения файла: выполнить команду меню **Файл | Сохранить,** указав в открывшемся окне навигации по файловой системе папку для сохранения скетча и его название, и нажать кнопку **OK**. Можно вместо команды **Файл | Сохранить** просто нажать кнопку **Сохранить** на панели инструментов (она там самая правая). При последующих сохранениях модифицированного (но уже сохраненного однажды) скетча папку и название скетча больше указывать не нужно. Но если нужно сохранить скетч в другую папку или под другим названием, тогда можно использовать опцию **Файл | Сохранить как** и указать в открывшемся окне навигации по файловой системе новую папку и/или название скетча.

Обсуждение работы скетча и возможных проблем

При сохранении файла скетча в среде Arduino IDE открывается стандартное системное окно навигации по файловой системе. В нем по умолчанию предлагается сохранить скетч в папке Arduino папки Мои документы (или в папке Документы на компьютерах Mac). Предлагаемое по умолчанию название скетча можно изменить на другое, которое более точно отражает предназначение скетча. Чтобы сохранить скетч, нажмите кнопку **OK**.

Название по умолчанию скетча состоит из слова *sketch*, затем символа подчеркивания, затем названия текущего месяца и даты и, наконец, буквы английского алфавита, начиная с *a*. Например, *sketch_may17b*. Использование значащего названия скетча вместо названия по умолчанию поможет определить назначение скетча при последующем обращении к нему.

Если в названии файла скетча указать запрещенный символ (например, пробел), среда Arduino IDE заменит его символом подчеркивания.

Скетчи Arduino сохраняются в простых текстовых файлах, но с расширением ino. В более старых версиях среды Arduino IDE файлам назначалось расширение pde, которое также используется для файлов скетчей языка программирования Processing. Скетчи автоматически сохраняются в папке с таким же названием, что и у сохраняемого скетча.

Скетчи можно сохранять в любой папке, но если сохранять их в папке по умолчанию (папка Arduino папки Мои документы в Windows или ~/Arduino в Linux), то сохраненные скетчи будут отображаться в меню **Папка со скетчами** меню **Файл** среды Arduino IDE, что облегчит их поиск.

Модифицированные скетчи примеров нельзя сохранять в исходной папке примеров, и при попытке сохранить такой скетч отобразится диалоговое окно с соответ-

ствующим предупреждением, а после нажатия в этом окне кнопки **OK** откроется окно навигации по файловой системе для выбора папки, в которой вы сможете сохранить модифицированный файл примера.

При попытке закрыть модифицированный пользовательский скетч откроется диалоговое окно с запросом, сохранить ли выполненные изменения. Если скетч содержит несохраненные изменения, на это укажет символ $ после названия скетча, которое отображается на вкладке вверху области редактирования. После сохранения скетча этот символ удаляется.

По мере разработки и модифицирования скетча может потребоваться следить за вносимыми в него изменениями. Самый легкий способ решить эту задачу — воспользоваться системой управления версиями Git (информацию по ее установке вы найдете на странице **Install Git** веб-сайта Atlassian Git Tutorial по адресу: **https://oreil.ly/0wJJC**). Работа с системой Git обычно осуществляется через интерфейс командной строки, но и имеются также и графические клиенты. Далее приводится базовая процедура для размещения скетча под управление версиями посредством системы Git:

1. Определите, в какой папке находится требуемый скетч. Это можно сделать, выполнив команду меню **Скетч | Показать папку скетча**. В результате выполнения этой команды папка скетча откроется в окне диспетчера файлов.
2. Запустите программу интерфейса командной строки в Windows или программу терминала в Linux или macOS. С помощью команды `cd` перейдите в папку скетча. Например, если скетч Blink находится в папке скетчей по умолчанию, переход в эту папку осуществляется следующим образом в macOS, Linux и Windows, соответственно:

```
$ cd ~/Documents/Arduino/Blink
$ cd ~/Arduino/Blink
> cd %USERPROFILE%\Documents\Arduino\Blink
```

3. Выполните команду `git init`, чтобы инициализировать репозиторий Git.
4. Добавьте файл скетча в репозиторий Git, выполнив команду:

```
git add Blink.ini
```

(замените здесь *Blink.ino* названием добавляемого файла). Если кроме файла скетча папка скетча содержит другие файлы, их также нужно добавить в репозиторий командой:

```
git add название_файла
```

5. Произведя значительные модификации скетча, зафиксируйте их, выполнив команду:

```
git commit -a -m "описание изменений"
```

6. Само собой понятно, что вместо текста "*описание изменений*" добавляется описание изменений, выполненных в скетче.

После фиксации изменения в репозитории Git историю изменений можно просмотреть в журнале репозитория, выполнив команду:

```
git log
```

Каждой зафиксированной модификации присваивается хеш-значение, как показано выделением полужирным шрифтом в следующем примере:

```
commit 87e962e54fe46d9e2a00575f7f0d1db6b900662a (HEAD -> master)
Author: Brian Jepson <bjepson@gmail.com>
Date: Tue Jan 14 20:58:56 2020 -0500

    Внесены значительные улучшения

commit 0ae1a1bcb0cd245ca9427352fc3298d6ccb91cef (HEAD -> master)
Author: Brian Jepson <bjepson@gmail.com>
Date: Tue Jan 14 20:56:45 2020 -0500

    Описание изменений
```

С помощью этих хеш-значений можно работать с более старыми версиями файла. (При этом не обязательно использовать полное хеш-значение — части его, позволяющей различить разные версии файла, будет достаточно.) Восстановить одну из предыдущих версий файла можно с помощью команды:

```
git checkout хеш_значение название_файла
```

например:

```
git checkout 0ae1 Blink.ino
```

Также можно сравнить версии скетчей с помощью команды:

```
git diff хеш_значение_А.. хеш_значение_Б
```

например:

```
git diff 0ae1..7018
```

Полная документация по системе управления версиями Git доступна по адресу **https://git-scm.com/doc**.

Лучше всего проверять отсутствие синтаксических ошибок во вносимых в код скетча изменениях выполнением частой компиляции скетча. Так легче обнаружить местонахождение ошибок, поскольку они обычно связаны с внесенными изменениями.

Код, загруженный в плату Arduino, нельзя загрузить обратно в компьютер.

Поэтому обязательно сохраните любые изменения скетчей, которые могут вам потребоваться в дальнейшем.

Если попытаться сохранить файл скетча в папке с названием иным, чем название скетча, среда Arduino IDE сообщит о невозможности открыть такую папку и предложит нажать кнопку **OK**, чтоб создать папку с таким же названием, как и у скетча.

Скетчи должны сохраняться в папке с таким же названием, как и название скетча. При сохранении нового скетча среда Arduino IDE автоматически создает папку с названием таким же, как и название сохраняемого скетча.

Файлы скетчей, созданных в более ранних версиях среды разработки Arduino IDE, имеют другое расширение (*pde*), чем скетчи современной версии среды. Современная версия среды Arduino IDE может открывать такие файлы, но при сохранении создает файл с современным расширением: *ino*. Код скетчей, созданный в версиях среды Arduino IDE более ранних, чем версия 1.0, может не компилироваться в этой и более поздних версиях среды. Но большинство модификаций, которые нужно внести в старый код, чтобы он мог компилироваться, не представляют больших сложностей.

1.6. Простой первый проект Arduino

ЗАДАЧА

Создать легкий и занимательный проект Arduino.

РЕШЕНИЕ

В этом примере вы сможете получить представление о некоторых способах разработки проектов, которые рассматриваются более подробно в последующих главах.

Предлагаемый скетч основан на скетче мигания светодиодом из предыдущего примера, но вместо фиксированного периода задержки частота мигания светодиода в нем определяется значением выходного сигнала фоторезистора (см. *разд. 6.3*). Подключите фоторезистор к контактам своей платы Arduino, как показано на рис. 1.7.

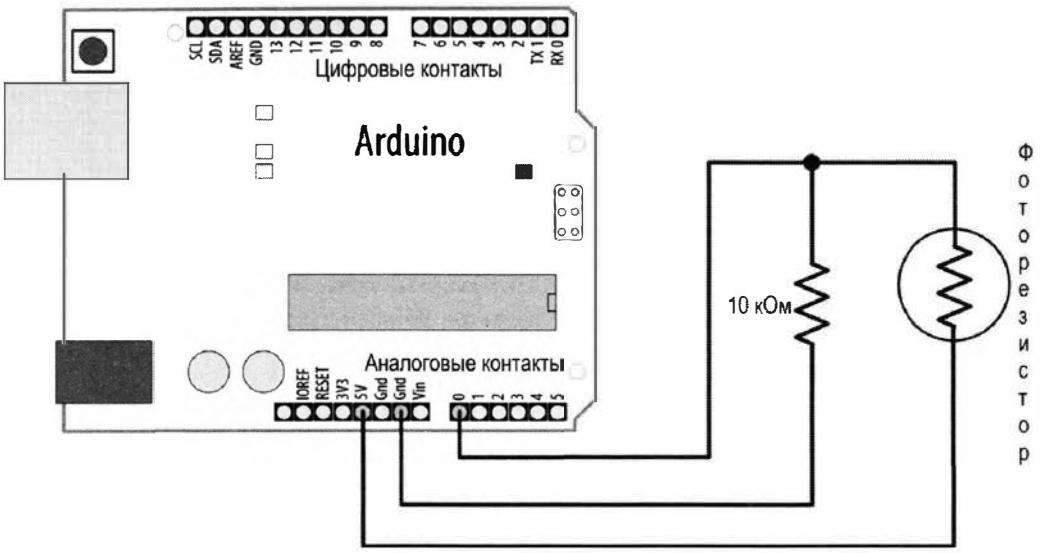


Рис. 1.7. Схема подключения фоторезистора к плате Arduino

Если у вас нет опыта сборки схем, руководствуясь принципиальной схемой, в *приложении 2* приводятся пошаговые инструкции по сборке такой схемы на макетной плате.

Фоторезисторы содержат опасное соединение — сульфид кадмия, поэтому они запрещены в некоторых странах. Если вы живете в одной из таких стран, то вместо фоторезистора можно использовать фототранзистор. У фототранзистора, так же как и у фоторезистора, один вывод более длинный, чем другой. Фототранзистор подключается так же, как и фоторезистор, но его длинный вывод подключается к контакту 5 В, а короткий — к контакту 0. Разные фототранзисторы реагируют на разные участки оптического спектра. Нам нужен фототранзистор, реагирующий на видимый свет (а не на инфракрасный) — такой как, например, фоторезистор компании Adafruit с артикулом 2831 (**https://oreil.ly/24xzl**).

Следующий далее скетч (листинг 1.2) считывает аналоговый сигнал, создаваемый подключенным к контакту 0 фоторезистором под воздействием освещения. Частота мигания светодиода будет меняться в зависимости от уровня освещения фоторезистора.

Листинг 1.2. Управление частотой мигания светодиода

```
/*
* Скетч управления скоростью мигания светодиода с помощью фоторезистора
*/

const int sensorPin = A0;         // Подключаем выход светодатчика
                                  // к контакту 0 аналогового ввода

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT); // Задаем выходной режим работы
                                  // для контакта подключения светодиода
}

void loop()
{
   int delayval = analogRead(sensorPin);  // Считываем входной аналоговый сигнал
   digitalWrite(LED_BUILTIN, HIGH);    // Включаем встроенный светодиод
   delay(delayval);                    // Длительность задержки
                                       // задается уровнем освещения
   digitalWrite(LED_BUILTIN, LOW);     // Выключаем встроенный светодиод
   delay(delayval);
}
```

В коде этого примера и в остальных примерах книги используется выражение `const int` для присвоения константам значащих названий (`sensorPin` — в нашем случае) вместо простых номеров (0, как могло бы быть). Использование констант рассматривается более подробно в *разд. 17.5*.

Обсуждение работы скетча и возможных проблем

Номинал сопротивления резистора в схеме на рис. 1.7 зависит от диапазона реагирования используемого вами фоторезистора. В частности, он должен быть приблизительно равным максимальному сопротивлению фоторезистора (т. е. сопротивлению неосвещенного фоторезистора). Это сопротивление можно определить, закрыв фоторезистор и измерив его мультиметром. Таким образом, если максимальное сопротивление фоторезистора составляет, например, 10 кОм, используйте резистор номиналом 10 кОм. Если же вместо фоторезистора используется фототранзистор, то сопротивление резистора может быть в диапазоне от 1 до 10 кОм. Выходной аналоговый сигнал фоторезистора, подаваемый на контакт 0 платы Arduino, будет изменяться в зависимости от уровня освещения фоторезистора. Функция `analogRead()` (она рассматривается более подробно в *главе 6*) считывает подаваемый на контакт 0 аналоговый сигнал и возвращает значения в диапазоне от приблизительно 200 для неосвещенного датчика до 800 при очень ярком его освещении. Этот диапазон возвращаемых функцией значений может варьироваться в зависимости от используемого фоторезистора и резистора, а также в зависимости от того, используется ли фототранзистор вместо фоторезистора. Возвращаемое функцией `analogRead()` значение определяет длительность периодов включенного и выключенного состояний светодиода, при этом такой период увеличивается с повышением освещенности.

Частоту мигания можно масштабировать, используя функцию `map()`, как показано в листинге 1.3.

Листинг 1.3. Масштабирование частоты мигания светодиода

```
/*
* Скетч управления частотой мигания светодиода
  с помощью фоторезистора с масштабированной частотой мигания
*/

const int sensorPin = A0;  // Подключаем выход светодатчика
                           // к контакту 0 аналогового ввода

// Предельные значения, возвращаемые функцией analogRead();
// Могут нуждаться в корректировке в зависимости от
// рассмотренных ранее в тексте факторов
const int low = 200;
const int high = 800;

// Следующие две строки кода задают максимальную и минимальную
// длительность паузы между миганиями.
const int minDuration = 100;   // Минимальная задержка между миганиями
const int maxDuration = 1000;  // Максимальная задержка между миганиями
```

```
void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);  // Задаем выходной режим работы
                                  // для контакта подключения светодиода
}

void loop()
{
   int delayval = analogRead(sensorPin);    // Считываем входной аналоговый сигнал

   // Следующие строки кода масштабируют значение задержки
   // между минимальным и максимальным значениями
   delayval = map(delayval, low, high, minDuration, maxDuration);
   delayval = constrain(delayval, minDuration, maxDuration);
   digitalWrite(LED_BUILTIN, HIGH);     // Включаем встроенный светодиод
   delay(delayval);                     // Длительность задержки
                                        // задается уровнем освещения
   digitalWrite(LED_BUILTIN, LOW);      // Выключаем встроенный светодиод
   delay(delayval);
}
```

Если изменение уровня освещенности фототранзистора не вызывает никаких изменений в частоте мигания светодиода, нужно поэкспериментировать с разными предельными выходными значениями констант `low` и `high`. В случае использования фототранзистора, попробуйте установить значение `low`, равное 10.

Использование функции `map()` для масштабирования значений рассматривается более подробно в *разд. 5.7*. А в *разд. 3.5* приводятся подробности использования функции `constrain()`, чтобы предельные значения не выходили за пределы заданного диапазона. Если по какой-либо причине будет получено значение задержки вне пределов диапазона, установленных значениями `low` и `high`, функция `map()` возвратит значение вне диапазона, определяемого значениями `minDuration` и `maxDuration`. Эта проблема со значениями, выходящими за пределы диапазона, решается вызовом функции `constrain()` после вызова функции `map()`, как и демонстрируется в коде листинга 1.3.

Значение переменной задержки `delayval` можно наблюдать на компьютере, выводя его в окно монитора порта среды Arduino IDE, как показано в следующем скетче (листинг 1.4). Окно монитора порта можно открыть, щелкнув на значке лупы (отдельный значок в правом конце панели инструментов среды Arduino IDE). Работа с монитором порта рассматривается более подробно в *главе 4*.

Листинг 1.4. Управление частотой мигания светодиода с помощью фоторезистора с масштабированной частотой мигания и выводом в монитор порта

```
/*
* Скетч управления частотой мигания светодиода с помощью фоторезистора
  с масштабированной частотой мигания и выводом в монитор порта.
*/
```

```
const int sensorPin = A0; // Подключаем выход светодатчика
                          // к контакту 0 аналогового ввода

// Предельные значения, возвращаемые функцией analogRead();
// Могут нуждаться в корректировке в зависимости
// от рассмотренных ранее в тексте факторов
const int low = 200;
const int high = 800;

// Следующие две строки кода задают максимальную и минимальную
// длительность паузы между миганиями
const int minDuration = 100;   // Минимальная задержка между миганиями
const int maxDuration = 1000;  // Максимальная задержка между миганиями

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);    // Задаем выходной режим работы
                                    // для контакта подключения светодиода
   Serial.begin(9600);              // Инициализируем библиотеку
                                    // последовательного интерфейса
}

void loop()
{
   int delayval = analogRead(sensorPin); // Считываем входной аналоговый сигнал
   // Следующие строки кода масштабируют значение задержки
   // между минимальным и максимальным значениями
   delayval = map(delayval, low, high, minDuration, maxDuration);
   delayval = constrain(delayval, minDuration, maxDuration);

   Serial.println(delayval);        // Выводим значение задержки в окно монитора порта
   digitalWrite(LED_BUILTIN, HIGH); // Включаем встроенный светодиод
   delay(delayval);                 // Длительность задержки задается
                                    // уровнем освещения
   digitalWrite(LED_BUILTIN, LOW);  // Выключаем встроенный светодиод
   delay(delayval);
}
```

Варьируя уровень освещения фоторезистора, можно не только менять частоту мигания светодиода, но также и частоту звукового сигнала, воспроизводимого динамиком. Соответствующая принципиальная схема показана на рис. 1.8, а скетч приведен в листинге 1.5.

Частоту включения и выключения сигнала управления нужно повысить до частоты в звуковом спектре. Для этого нужно уменьшить минимальное и максимальное значения задержки, как показано в листинге 1.5.

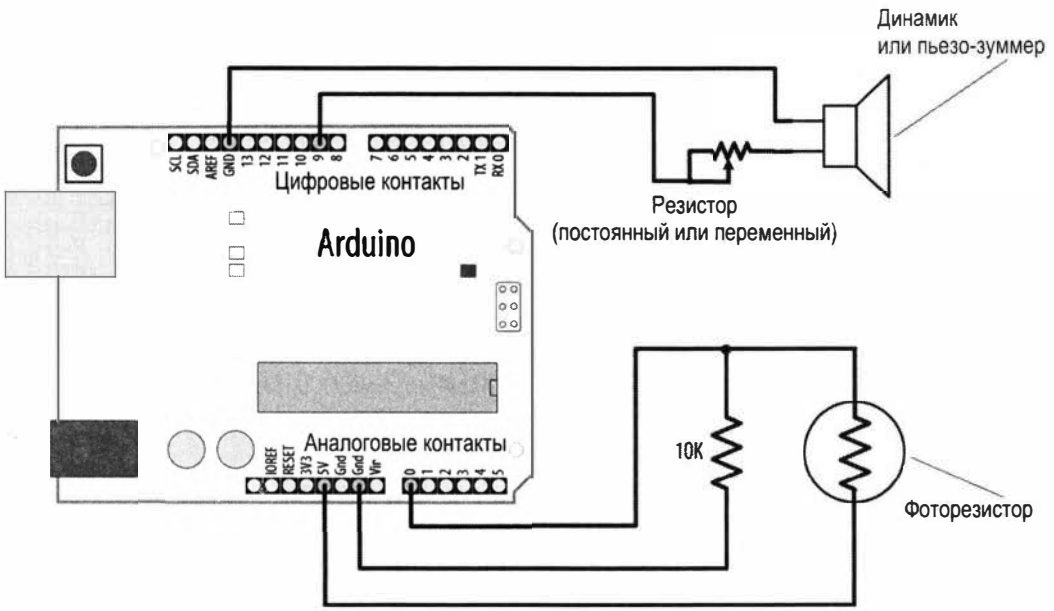


Рис. 1.8. Принципиальная схема подключения динамика

Листинг 1.5. Скетч для управления частотой воспроизводимого динамиком сигнала

```
/*
* Скетч для управления частотой воспроизводимого динамиком сигнала
*/

const int outputPin = 9;      // Подключаем один выход динамика
                              // к контакту 9 платы Arduino
const int sensorPin = A0;     // Подключаем выход светодатчика
                              // к контакту 0 аналогового ввода

const int low = 200;
const int high = 800;

const int minDuration = 1;    // Длительность включенного
                              // и выключенного состояний – 1 мс (500 Гц)
const int maxDuration = 10;   // Длительность включенного
                              // и выключенного состояний – 10 мс (50 Гц)

void setup()
{
   pinMode(outputPin, OUTPUT);   // Задаем выходной режим работы
                                 // для контакта подключения светодиода
}
```

```
void loop()
{
   int sensorReading = analogRead(sensorPin);  // Считываем входной аналоговый сигнал
   int delayval = map(sensorReading, low, high, minDuration, maxDuration);
   delayval = constrain(delayval, minDuration, maxDuration);

   digitalWrite(outputPin, HIGH);  // Устанавливаем на контакте
                                   // управления высокий логический уровень
   delay(delayval);     // Длительность задержки задается уровнем освещения
   digitalWrite(outputPin, LOW);   // Устанавливаем на контакте управления
                                   // низкий логический уровень
   delay(delayval);
}
```

Дополнительная информация

Подробно генерирование звуковых сигналов с помощью Arduino рассматривается в *главе 9*.

1.7. Работа с платами Arduino, неподдерживаемыми по умолчанию средой Arduino IDE

ЗАДАЧА

Для проекта нужно использовать плату, которая отсутствует в списке плат Arduino IDE — например, Arduino MKR 1010.

РЕШЕНИЕ

Чтобы в среде Arduino IDE можно было работать с платой, отсутствующей в списке плат по умолчанию, в среду IDE нужно добавить информацию о ней. Для этого надо открыть окно **Менеджер плат**, выполнив команду меню **Инструменты | Плата "Arduino/Genuino Uno" | Менеджер плат** (рис. 1.9).

Открывшееся окно Менеджера плат может не быть сразу доступным для работы в нем, поскольку среда Arduino IDE проверяет список определений плат, доступных в Интернете, чтобы обеспечить использование самых последних версий. Дождитесь, пока среда Arduino IDE не завершит эту проверку.

Добавление других плат в список плат Arduino IDE

Описанная далее процедура добавления в среду Arduino IDE платы Arduino MKR 1010 также применима и для других плат, которые нужно добавить в список доступных плат. Информацию о местонахождении файлов определения для конкретной платы можно найти в ее документации.

По завершении этой проверки в окне Менеджера плат (рис. 1.10) появится список уже установленных в среде Arduino IDE определений плат и определений, доступных для загрузки из Интернета.

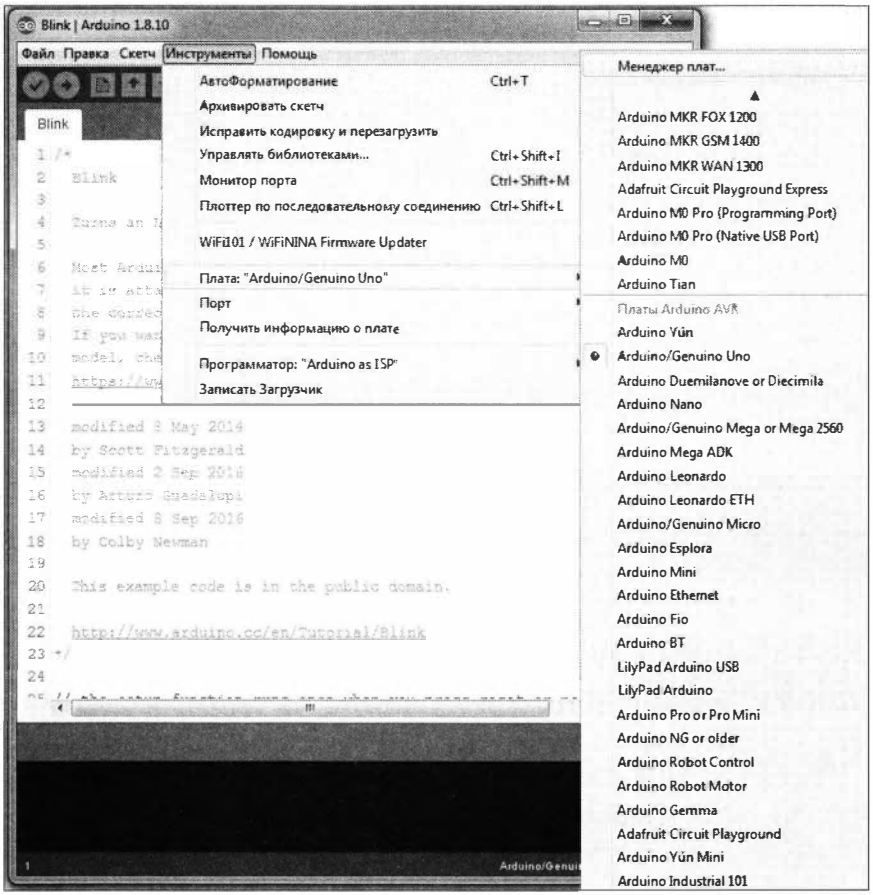


Рис. 1.9. Открытие окна **Менеджер плат** среды Arduino IDE

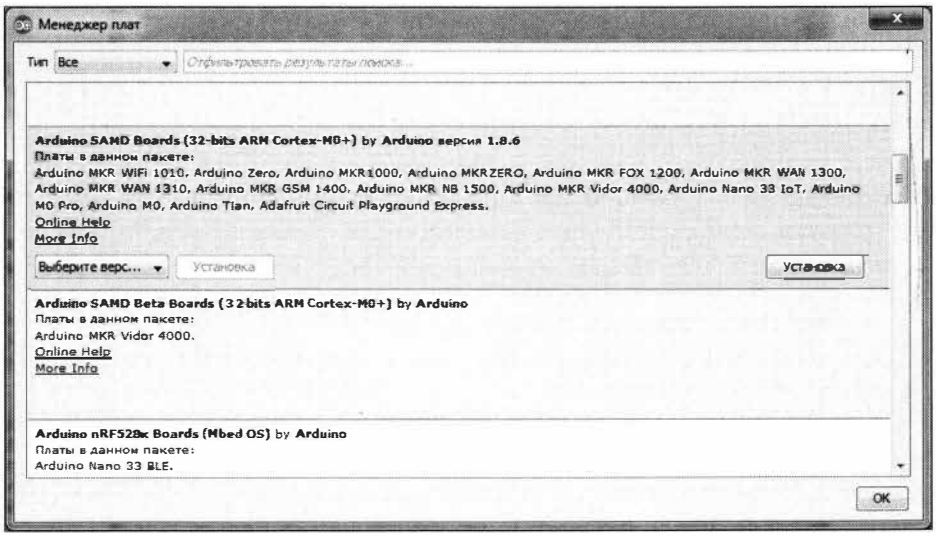


Рис. 1.10. Окно **Менеджер плат** среды Arduino IDE

Чтобы установить определение платы MKR 1010, нужно ввести ее название в строку поиска, в результате чего список будет отфильтрован до этого определения — **Arduino SAMD Board (32-bits ARM Cortex-M0+)**. Можно также прокрутить список вручную и самостоятельно найти это определение. В любом случае выберите требуемое определение, наведя на него курсор мыши[1], а затем нажмите отобразившуюся кнопку **Установить**.

Установка определения платы может занять некоторое время, поэтому наберитесь терпения.

Установив определение выбранной платы, можно продолжить установку определений для других плат или же закрыть окно Менеджера плат, если этого не требуется. Теперь список плат будет содержать опцию выбора для работы платы MRK 1010 (рис. 1.11).

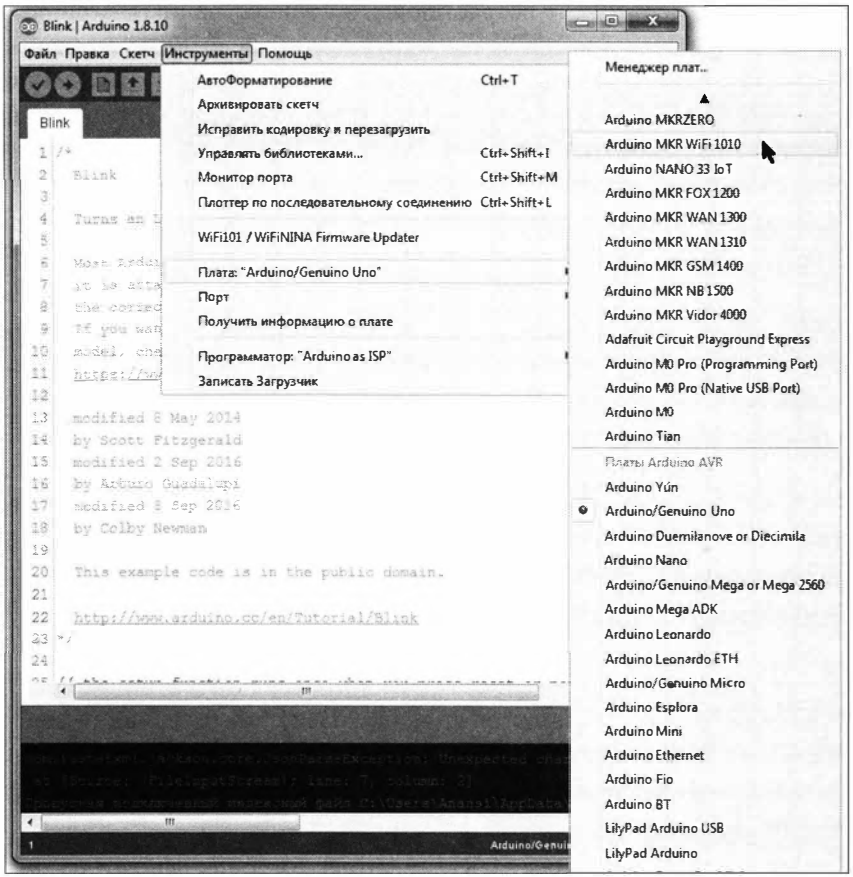


Рис. 1.11. После установки определения платы MRK 1010 она отображается в списке плат среды Arduino IDE

[1] В рассматриваемом случае отфильтрованный список будет содержать только одно определение платы, и выбирать ничего не придется. Но иногда он может содержать несколько разных определений, содержащих ключевое слово поиска. Тогда из этих определений надо выбрать нужное. То же самое относится и к ручному поиску определения платы.

Обсуждение работы решения и возможных проблем

Файлы, загружаемые при добавлении определения новой платы, содержат инструкции по сопоставлению программных компонентов среды Arduino IDE для работы с определенными аппаратными составляющими микроконтроллера платы, местонахождению этих составляющих в конкретном микроконтроллере или семействе микроконтроллеров.

Установленное в среду Arduino IDE описание для платы с определенным микроконтроллером часто позволяет работать с семейством плат, использующих этот микроконтроллер. Например, установка поддержки для платы MKR 1010 также позволяет работать с платой Arduino Zero, поскольку в обеих платах используется одинаковый микроконтроллер.

Чтобы облегчить поддержку все возрастающего количества плат Arduino и их клонов, в выпуск 1.6 среды Arduino IDE было добавлено средство для управления платами — Менеджер плат. С помощью этого средства процесс добавления поддержки плат в среду Arduino IDE и удаления ее стал гораздо проще. Оно также позволяет обновлять файлы поддержки в случае выхода их новых версий или же выбирать конкретную версию поддержки для определенной задачи. Среда Arduino IDE больше не содержит файлы поддержки для всех возможных плат Arduino, поэтому, даже установив самую последнюю версию этой среды, вы можете не обнаружить в ней установленную поддержку для своей платы.

Средство Менеджер плат позволяет производителям совместимых с Arduino плат добавлять описание своих плат в среду Arduino IDE. Если в Интернете доступно описание совместимой платы в правильном формате, интернет-адрес (URL) этого описания можно добавить в качестве одного из источников, используемых Менеджером плат для составления списка плат, которым можно установить поддержку. Это означает, что наличие в Интернете файлов поддержки для той или иной платы также станет проверяться при открытии окна Менеджера плат, поэтому пользователь будет извещаться о доступных для установки обновлениях с помощью того же самого механизма, что и для обновления файлов поддержки стандартных плат Arduino.

Процедура добавления интернет-адреса (URL) местонахождения файлов поддержки для совместимых плат Arduino следующая. Выполните команду меню **Файл | Настройки** и в открывшемся окне **Настройки** нажмите кнопку справа от поля **Дополнительные ссылки для Менеджера плат**, чтобы открыть одноименное диалоговое окно (рис. 1.12).

Если вам известен интернет-адрес (URL) местонахождения файлов поддержки для своей совместимой платы, вставьте его в поле ввода этого окна. Если окно уже содержит какие-либо адреса, новый адрес нужно вставлять в новою строку. Если же вы не знаете этот адрес, перейдите по ссылке **Нажмите для получения ресурсов для работы с неофициальными платами**, чтобы открыть веб-страницу со списком адресов местонахождения файлов поддержки неофициальных плат Arduino (**https://oreil.ly/Ceziq**) и попробуйте найти требуемую ссылку на этой странице.

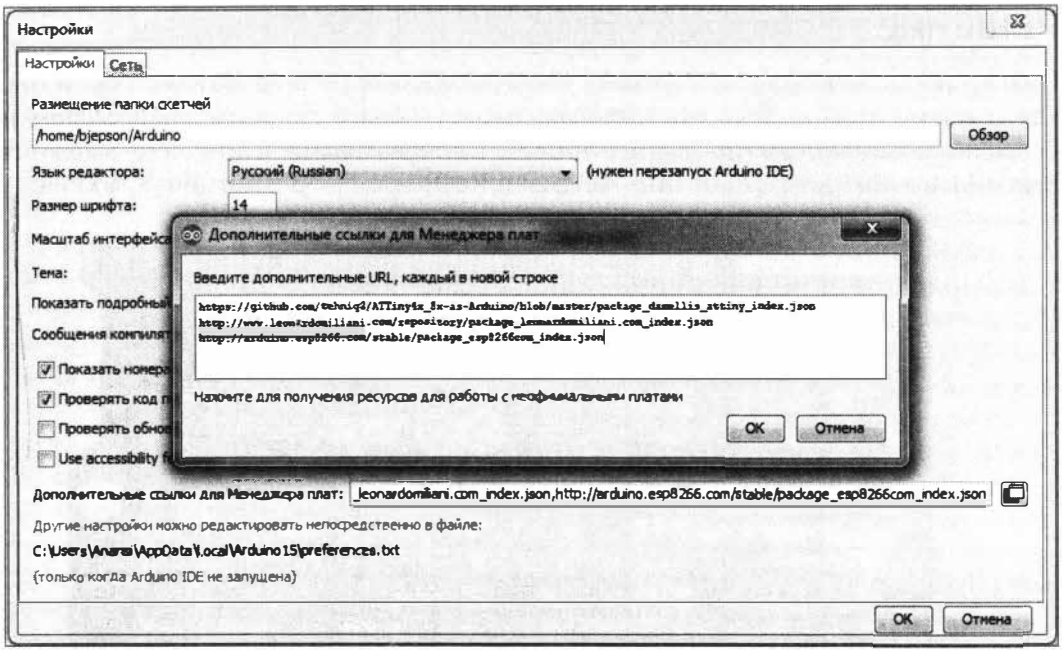


Рис. 1.12. Окно **Дополнительные ссылки для Менеджера плат**

Для установки поддержки некоторых плат эта процедура не подходит, и приходится устанавливать специальную поддержку, предоставляемую изготовителем. Например, чтобы установить поддержку для плат Teensy (**https://www.pjrc.com/teensy**), нужно загрузить отдельную программу установщика этой поддержки с веб-сайта компании Teensy (**https://oreil.ly/J_kfi**). При этом следует иметь в виду, что надо использовать такую версию установщика Teensy, которая поддерживается используемой вами версией среды разработки Arduino IDE. Компания обычно предоставляет версию установщика для новой версии среды Arduino IDE в течение одной-двух недель после ее выхода.

Дополнительная информация

Краткие руководства по использованию разных плат Arduino можно найти по этой ссылке: **https://oreil.ly/KyWcO**.

1.8. Работа с 32-разрядными (или совместимыми с ними) платами Arduino

ЗАДАЧА

Включить в работу плату форм-фактора Arduino Uno, но с 32-разрядной производительностью.

РЕШЕНИЕ

Плата Arduino Zero (рис. 1.13) имеет такие же размеры и организацию контактов, как и плата Arduino Uno, но оснащена намного более мощным процессором и большим объемом памяти. Совместимыми альтернативами плате Zero являются платы Metro M0 Express компании Adafruit и RedBoard Turbo компании SparkFun.

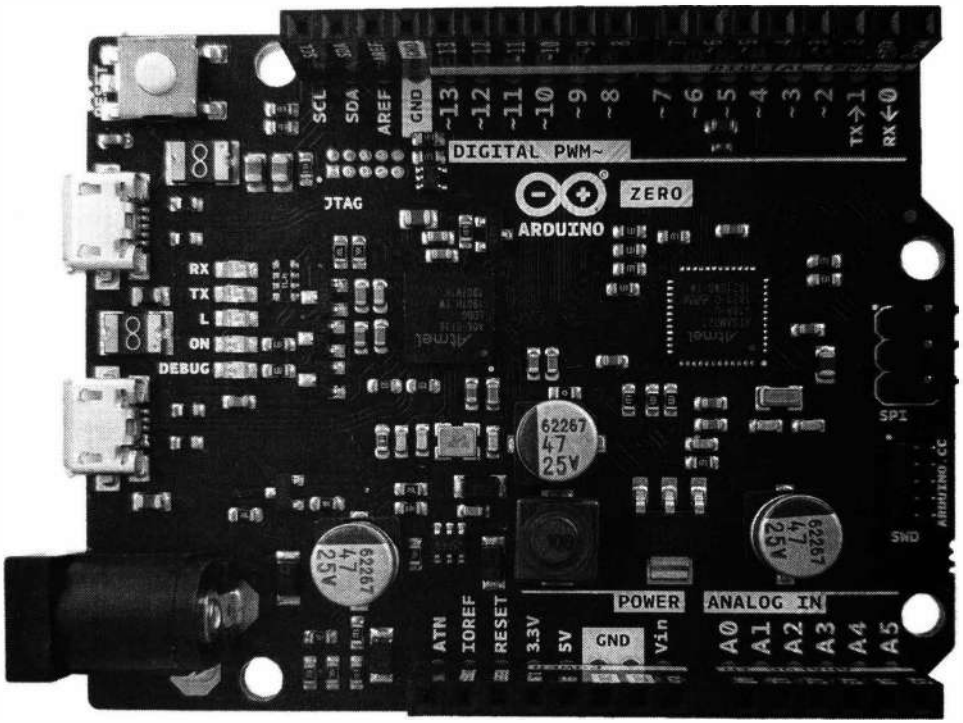


Рис. 1.13. Плата Arduino/Genuino Zero

Несмотря на то что плата Zero внешне очень похожа на плату Uno, между этими платами есть несколько важных различий. И самое важное из них — использование в плате Zero 32-разрядного микропроцессора архитектуры ARM (SAMD21 производства компании Microchip). Скетч в листинге 1.6 похож на скетч из предыдущего примера, и некоторые различия между ними связаны с рядом важных различий между платами на процессорах ARM и платой Uno.

Листинг 1.6. Скетч управления частотой выходного сигнала для платы Zero

```
/*
* Скетч управления частотой выходного сигнала для платы Zero
*/
const int outputPin = A0; // Подключаем наушники к аналоговому выходному контакту 0
const int sensorPin = A1; // Подключаем выход светодатчика
                          // к контакту 1 аналогового ввода
```

```
const int low = 200;
const int high = 800;

const int sampleCount = 16; // Количество выборок для вывода одного цикла

const int minDur = 1000000/(sampleCount*500); // Период в мкс для 500 Гц
const int maxDur = 1000000/(sampleCount*50);  // Период в мкс для 50 Гц

// Таблица значений для 16 выборок одного цикла синусоидального сигнала
static int sinewave[sampleCount] =
{
   0x1FF,0x2B6,0x355,0x3C7,0x3FF,0x3C7,0x355,0x2B6,
   0x1FF,0x148,0x0A9,0x037,0x000,0x037,0x0A9,0x148
};

void setup()
{
   analogWriteResolution(10); // Задаем максимальное разрешение
                              // для ЦАП платы Arduino
}

void loop()
{
   int sensorReading = analogRead(sensorPin); // Считываем входной аналоговый сигнал
   int duration = map(sensorReading, low, high, minDur, maxDur);
   duration = constrain(duration, minDur, maxDur);
   duration = constrain(duration, minDur, maxDur);

   for(int sample=0; sample < sampleCount; sample++)
   {
      analogWrite(outputPin, sinewave[sample]);
      delayMicroseconds(duration);
   }
}
```

Прежде чем загружать этот скетч в плату Arduino Zero, плату Metro M0 (или M4) компании Adafruit или плату RedBoard компании SparkFun, для этих плат нужно установить соответствующую поддержку в порядке, описанном в *разд. 1.7*. При этом для плат компаний Adafruit и SparkFun надо сначала указать в Менеджере плат интернет-адрес (URL) местонахождения этих файлов. Этот адрес можно взять со страниц: **https://oreil.ly/l2paC** и **https://oreil.ly/DLM0a** для плат Adafruit и SparkFun соответственно. После установки файлов поддержки для платы SAMD, выберите свою плату в списке плат в среде Arduino IDE, а также укажите правильный последовательный порт для ее подключения. Далее соберите на макетной плате схему, показанную на рис. 1.14, и загрузите скетч в плату.

Если изменение уровня освещенности фоторезистора не вызывает никаких изменений в частоте сигнала в наушниках, нужно поэкспериментировать с разными предельными выходными значениями констант `low` и `high`. В случае использования фототранзистора, попробуйте установить значение `low`, равное 10. Если освещение создается лампами дневного света или светодиодными лампами, поверх основного звукового сигнала может быть слышен своеобразный звук, наподобие стрекотания сверчка. Этот звук создается миганием этих источников света, невидимым для человеческого глаза.

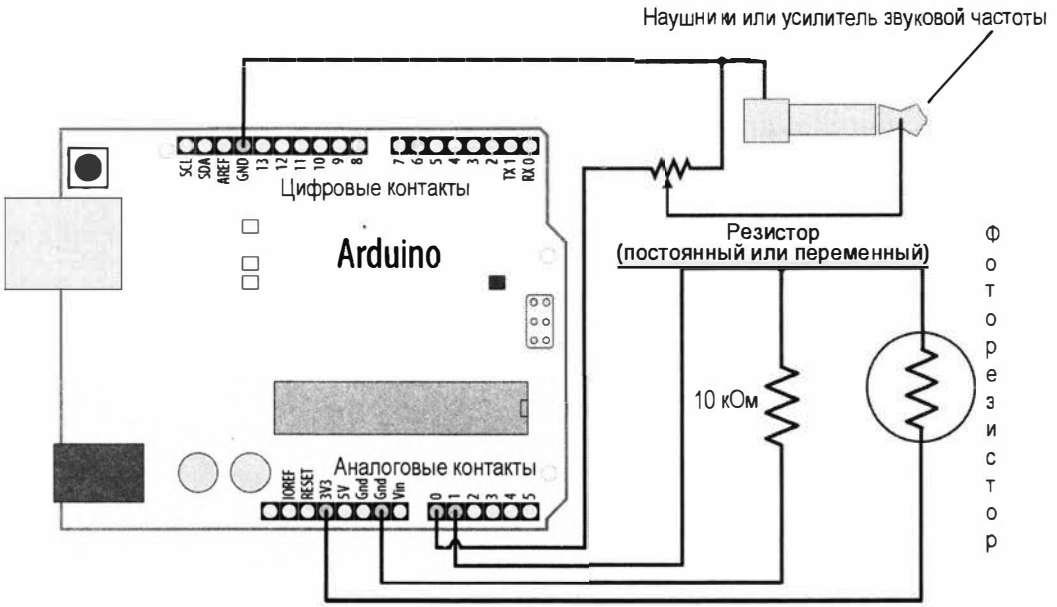


Рис. 1.14. Принципиальная схема для работы со скетчем из листинга 1.6

Платы на процессорах SAMD не могут работать с сигналами напряжением 5 В

На контакты ввода/вывода этих плат нельзя подавать сигналы напряжением выше 3,3 В, поскольку это может вывести плату из строя.

Обсуждение работы примера и возможных проблем

Хотя на первый взгляд схема, приведенная на рис. 1.14, может показаться похожей на схему из рис. 1.8, для входа датчика и аудиовыхода в них задействованы разные контакты. Платы Arduino Zero и совместимые с ними оснащены цифроаналоговым преобразователем (ЦАП), который может создавать более реалистичный звук, чем двоичный выход стандартных цифровых выводов. Но выходной аналоговый сигнал ЦАП подается только на аналоговый контакт 0, что делает этот контакт недоступным для подачи входного сигнала с фоторезистора. Соответственно этот входной сигнал подается на аналоговый контакт 1.

Другое различие, которое нельзя увидеть на схеме, состоит в том, что платы Arduino Zero и совместимые с ними могут выдавать на своих контактах ток силой всего лишь в 7 мА, что значительно ниже тока в 40 мА на контактах платы Uno. А поскольку максимальное напряжение на их контактах ограничено величиной 3,3 В (по сравнению с 5 В плат Uno), то максимальная мощность на контактах этих плат почти в десять раз меньше, чем на платах Uno. Вот поэтому мы не можем подать выходной сигнал с контакта непосредственно на динамик, а должны подавать его на наушники или на усилитель звуковой частоты. Мощности выходного сигнала просто недостаточно для генерирования звука при прямой подаче на динамик.

В скетче используется таблица поиска из 16 выборок на один цикл синусоидального сигнала, но эти платы обладают достаточной вычислительной мощностью, чтобы обеспечить намного более высокое разрешение. Поэтому количество выборок можно увеличить, чтобы улучшить качество звукового сигнала.

Дополнительная информация

Краткое руководство по использованию плат Arduino Zero можно найти здесь: **https://oreil.ly/JhTie**. Работа со звуком рассматривается более подробно в *главе 9*.

ГЛАВА 2

Программирование на языке Arduino

2.0. Введение

Хотя большая часть работы над проектами Arduino будет состоять в подключении необходимых устройств к плате Arduino, нам также нужно будет дать плате инструкции, что и как делать с этими устройствами. В этой главе представлены основные сведения по программированию на языке Arduino, приводятся наиболее распространенные конструктивные элементы языка для читателей, незнакомых с программированием вообще, а также дается обзор синтаксиса языка для тех читателей, кто не знаком с языком C/C++, на котором основан язык Arduino.

Чтобы сделать примеры этой главы более интересными (а это возможно только, когда их результаты реально ощутимы), в них используются физические возможности платы, которые подробно рассматриваются в последующих главах. И если вам что-либо непонятно в приведенном в этой главе коде, вы можете прояснить для себя эти моменты, обратившись к материалу следующих глав, — особенно к *главе 4*, где подробно рассматривается последовательный интерфейс, и к *главе 5*, освещающей использование цифровых и аналоговых контактов платы Arduino. Впрочем, чтобы разобраться с работой конкретных возможностей, на которых концентрируются решения этой главы, понимать весь их код совершенно необязательно. Далее приводится список и краткое описание некоторых из наиболее часто употребляемых функций, используемых в примерах этой главы, которые рассматриваются более подробно в последующих главах:

- `Serial.println(value);`

  Выводит указанное в параметре значение в окно монитора порта среды Arduino IDE, что позволяет просматривать вывод с платы Arduino на своем компьютере. Более подробно последовательная связь рассматривается в *главе 4*, а вывод с платы Arduino в монитор порта — в *разд. 4.1*.

- `pinMode(pin, mode);`

  Конфигурирует цифровой контакт платы Arduino для работы в режиме ввода (чтение с внешнего устройства) или вывода (запись на внешнее устройство) значений. Более подробно рассматривается во *введении* в *главу 5* (*разд. 5.0*).

- `digitalRead(pin);`

  Считывает входной цифровой сигнал с контакта, работающего в режиме ввода. Дополнительная информация приводится в *разд. 5.1*.

♦ `digitalWrite(pin, value);`

Подает выходной цифровой сигнал на контакт, работающий в режиме вывода. Дополнительная информация приводится в *разд. 5.1*.

2.1. Типичный скетч Arduino

ЗАДАЧА

Получить понимание базовой структуры скетча (программы) на языке Arduino. Эта структура демонстрируется в скетче в листинге 2.2, который управляет платой Arduino, мигая подключенным к ней светодиодом.

РЕШЕНИЕ

Программы для управления функциональностями платы Arduino обычно называются *скетчами*, от английского слова «sketch», которое переводится на русский язык как «набросок». Изначально пользователями платформы Arduino были художники и дизайнеры, которые в своей работе используют наброски, чтобы легко и быстро реализовать какую-либо пришедшую им в голову идею. Термины «скетч» и «программа» равнозначны. Скетч содержит исходный код — инструкции для выполнения платой Arduino тех или иных действий. Код для действий, которые нужно выполнить только один раз (например, выполнить конфигурирование платы для осуществления определенных действий), размещается в функции `setup()`. А код для действий, которые нужно исполнять в бесконечном цикле после завершения конфигурации платы, размещается в функции `loop()`. В листинге 2.1 демонстрируется организация типичного скетча Arduino

Листинг 2.1. Организация типичного скетча Arduino

```
// Функция setup() исполняется только один раз при запуске скетча
void setup()
{
   pinMode(LED_BUILTIN, OUTPUT); // Задает выходной режим работы
                                 // контакта, к которому подключен встроенный светодиод
}

// Функция loop() исполняется в бесконечном цикле
void loop()
{
   digitalWrite(LED_BUILTIN, HIGH); // Включаем светодиод
   delay(1000); // wait a second
   digitalWrite(LED_BUILTIN, LOW); // Выключаем светодиод
   delay(1000);                    // Пауза длительностью в 1 секунду
}
```

Инструкции в скетче Arduino начинают исполняться по завершении загрузки кода в плату Arduino или после подачи питания на плату и исполняются последовательно сверху вниз. Инструкции в функции `setup()` исполняются один раз, а затем начинают исполняться инструкции в функции `loop()`. После исполнения последней инструкции функции `loop()` (которая обозначена следующей за ней закрывающей фигурной скобкой `}`) функция `loop()` вызывается снова и процесс повторяется бесконечно, пока не будет отключено питание платы или нажата кнопка сброса.

Обсуждение работы решения и возможных проблем

Только что рассмотренный пример скетча постоянно мигает встроенным в плату светодиодом, попеременно подавая на контакт, к которому подключен светодиод, высокий и низкий уровни напряжения (работа с контактами платы Arduino рассматривается более подробно в *главе 5*). Когда скетч начинает исполняться, код в функции `setup()` конфигурирует контакт, к которому подключен встроенный светодиод (`LED_BUILTIN`) для работы в режиме вывода (`OUTPUT`), чтобы он мог включать и выключать этот светодиод. После завершения исполнения кода в функции `setup()` код в функции `loop()` (который мигает светодиодом) исполняется в бесконечном цикле, пока на плату Arduino подается питание.

Следующая информация не требуется для разработки скетчей Arduino, но может быть интересной. Как уже упоминалось, язык Arduino основан на языке C/C++, поэтому опытные программисты могут задаваться вопросом, куда же подевалась функция входа в программу `main()`, которая является необходимой функцией всех программ на языке C/C++. Она никуда не делась и присутствует во всех скомпилированных скетчах Arduino, но скрыта от пользователя средой сборки Arduino. Процесс сборки создает промежуточный файл, содержащий код скетча, а также дополнительные операторы. В листинге 2.2 показано, как выглядит функция `main()` для 8-разрядных плат (для 32-разрядных плат используется подобная функция).

Листинг 2.2. Структура функции `main()` для 8-разрядных плат Arduino

```
int main( void )
{
   init();

   initVariant();

#if defined(USBCON)
   USBDevice.attach();
#endif

   setup();

   for (;;)
   {
      loop();
```

```
        if (serialEventRun) serialEventRun();
    }

    return 0;
}
```

Первым делом код функции `main()` вызывает функцию `init()`, которая выполняет инициализацию аппаратного обеспечения платы Arduino. Затем вызывается функция `initVariant()`. Это очень редко используемая точка входа в программу, которая предоставляет разработчикам совместимых с Arduino плат способ вызова собственных специализированных процедур для инициализации платы. Если микроконтроллер платы содержит выделенное оборудование для поддержки USB, оно подготавливается к работе (вызовом функции `attach()`).

Далее вызывается функция `setup()` собственно скетча, после которой функция `loop()` скетча вызывается в бесконечном цикле `for`. Поскольку цикл `for` никогда не завершается, оператор `return` никогда не исполняется.

Непосредственно перед вызовом функции `loop()` вызывается функция `serialEventRun()`, если она поддерживается платой (платы с микроконтроллером ATmega32U4 — например, Leonardo, не поддерживают эту функцию). Это позволяет использовать в скетче специальную функцию `serialEvent()`, которая вызывается, когда на последовательном порту появляются входные данные (см. *разд. 4.3*).

Дополнительная информация

Процесс загрузки скетча в плату Arduino подробно описан в *разд. 1.4*. Более подробная информация по процессу сборки скетчей Arduino приводится в *главе 17* и на веб-странице **Sketch build process (https://oreil.ly/c8YQW)**.

2.2. Простые типы данных (переменные)

ЗАДАЧА

В языке Arduino для представления разных значений используются переменные разных типов данных. Мы хотим знать, как выбрать и использовать эти типы данных Arduino для конкретных задач.

РЕШЕНИЕ

Для большинства приложений Arduino наиболее часто используемым типом данных для числовых значений является тип `int` (сокращение от integer, целое число). Но если по какой-либо причине этот тип данных не подходит для конкретной задачи, определить тип данных, который наиболее подходит для работы с диапазоном значений, ожидаемых приложением, можно по табл. 2.1 и 2.2 (в табл. 2.1 показаны типы данных для 8-разрядных плат, а в табл. 2.2 — для 32-разрядных).

Таблица 2.1. *Типы данных языка Arduino для 8-разрядных плат*

Тип данных	Кол-во байтов	Диапазон значений	Применение
Основные типы			
`int`	2	–32768 до 32767	Представляет положительные и отрицательные целочисленные значения
`unsigned int`	2	0 до 65535	Представляет только положительные значения; в остальном то же самое, что и `int`
`long`	4	–2147483648 до 2147483647	Представляет очень большой диапазон положительных и отрицательных значений
`unsigned long`	4	4294967295	Представляет очень большой диапазон положительных значений
`float`	4	3,4028235E+38 до –3,4028235E+38	Представляет числа с дробной частью; используется для аппроксимирования реальных измерений
`double`	4	То же самое, что и `float`	В языке Arduino тип данных `double` — просто другое название типа данных `float`
`bool`	1	Ложь (0) или истина (1)	Представляет значения `true` (истина) и `false` (ложь)
`char`	1	–128 до 127	Представляет один символ. Может также представлять числовое значение со знаком в диапазоне от –128 до 127
`byte`	1	0 до 255	Подобен типу `char`, но для значений без знака
Прочие типы			
`String`			Представляет последовательность символов, обычно содержащую текст
`void`			Применяется только для объявления функций, которые не возвращают значения

Таблица 2.2. *Типы данных языка Arduino для 32-разрядных плат*

Тип данных	Кол-во байтов	Диапазон значений	Применение
Основные типы			
`short int`	2	–32768 до 32767	То же самое, что и тип `int` для 8-разрядных плат
`unsigned short int`	2	0 до 65535	То же самое, что и беззнаковый тип `unsigned int` для 8-разрядных плат
`int`	4	–2147483648 до 2147483647	Представляет положительные и отрицательные целочисленные значения
`unsigned int`	4	0 до 4294967295	Представляет только положительные значения; в остальном то же самое, что и `int`

***Таблица 2.2** (окончание)*

Тип данных	Кол-во байтов	Диапазон значений	Применение
`long`	4	–2147483648 до 2147483647	То же самое, что и `int`
`unsigned long`	4	4294967295	То же самое, что и `unsigned int`
`float`	4	±3,4028235E+38	Представляет числа с дробной частью; используется для аппроксимирования реальных измерений
`double`	8	±1,7976931348623158E+308	32-разрядные платы поддерживают дробные числа значительно большего диапазона и точности, чем 8-разрядные платы
`bool`	1	ложь (0) или истина (1)	Представляет истинные и ложные значения
`char`	1	–128 до 127	Представляет один символ. Может также представлять числовое значение со знаком в диапазоне от –128 до 127
`byte`	1	0 до 255	Подобен типу `char`, но для значений без знака
Прочие типы			
`String`			Представляет последовательность символов, обычно содержащую текст
`void`			Применяется только для объявления функций, которые не возвращают значения

Обсуждение работы решения и возможных проблем

Для числовых переменных, величина значений которых не выходит за пределы указанного в табл. 2.1 диапазона, в большинстве случаев подойдет целочисленный тип данных `int`. Исключение могут составлять ситуации, требующие максимальной производительности или наиболее эффективного использования памяти. В большинстве официальных примеров кода на Arduino присутствуют переменные типа `int`. Но иногда определенные задачи требуют применения специфичного типа данных. Этот момент имеет особенную важность при вызове библиотек, возвращающих значения иного типа, чем тип `int`. Например, функция `millis()`, подробно рассматриваемая в *разд. 12.1* и задействованная во многих примерах этой книги, возвращает беззнаковое длинное целочисленное значение типа `unsigned long`. Если на 8-разрядной плате сохранить результаты этой функции в переменную типа `int`, компилятор не выдаст никакого предупреждения, но будет сохранено неправильное значение, поскольку переменная типа `int` недостаточно большая, чтобы в ней можно было сохранить максимальное значение типа `long`. В частности, значения, превышающие 32 767, будут сохраняться как нарастающие отрицательные значения,

начиная с –32 768. А при попытке сохранить значение типа `long` в переменную типа `unsigned int` произойдет переход к началу отсчета (к 0) после превышения максимального значения для типа `unsigned int` (65,535).

В одних случаях для решения задачи требуется использовать отрицательные числа, а в других нет, поэтому язык Arduino поддерживает два вида числовых типов: `signed` (со знаком) и `unsigned` (без знака, или беззнаковые). Беззнаковые значения всегда положительные. Типы данных, указанные без ключевого слова `unsigned`, являются знаковыми и могут представлять как положительные, так и отрицательные значения. Одна из причин для использования беззнаковых значений возникает, когда диапазон значений знакового типа меньше диапазона значений переменной (диапазон значений переменной беззнакового типа вдвое больше диапазона значений переменной знакового типа). Другой причиной, по которой программисты используют беззнаковые переменные, является ситуация, когда они хотят явно дать понять пользователям кода, что ожидаемое значение этой переменной никогда не будет отрицательным.

На 32-разрядных платах переменная типа `int` требует вдвое больше байтов, чем такая же переменная на 8-разрядных платах, но поскольку 32-разрядные платы оснащены большими объемами памяти, то большинство кода для 8-разрядных плат будет без проблем исполняться на 32-разрядных. Редкое исключение возникает в том случае, когда код предполагает, что переменные типа `int` всегда будут храниться в двух байтах памяти. Но это нечто такое, что качественный код прикладных программ и библиотек никогда не должен делать.

Переменные типа `bool` (булевы) могут иметь только два значения: `true` (истина) и `false` (ложь). Такие переменные обычно служат для хранения значений, представляющих состояние да/нет. Переменные булева типа также могут использоваться вместо встроенных констант `HIGH` и `LOW`, которые модифицируют или определяют состояние цифровых контактов ввода/вывода (посредством функций `digitalWrite()` и `digitalRead()` соответственно). Например, функция:

```
digitalWrite(LED_BUILTIN, HIGH);
```

подает сигнал высокого уровня на контакт, к которому подключен встроенный светодиод, включая его. Если вместо `HIGH` вторым параметром этой функции указать `LOW`, на контакт будет подаваться сигнал низкого уровня, что выключит светодиод. Вместо параметров `HIGH` и `LOW` можно использовать параметры `true` и `false` с тем же самым эффектом. В Интернете можно найти много примеров кода с таким использованием булевых переменных. Также можно найти примеры, в которых используются числовые значения 1 и 0, представляя булевы значения `true` и `false` соответственно. Однако подобные решения не приветствуются, поэтому всегда следует использовать константы `HIGH` и `LOW`. Вероятность ситуации, в которой константа `HIGH` была бы равнозначной булевому значению `false`, чрезвычайно низкая. Но очень вероятно, что вам могут повстречаться многие другие константы, большинство из которых не имеют такой явной и очевидной взаимосвязи с их базовыми значениями.

Дополнительная информация

Подробная информация по типам данных приводится на странице справки вебсайта Arduino (**https://oreil.ly/xRLxx**).

2.3. Работа с числами с плавающей запятой (точкой)

ЗАДАЧА

Числа с плавающей запятой (точкой[1]) используются для выражения значений с десятичной запятой (таким способом представляются дробные значения). В скетче требуется выполнять вычисления и сравнивать такие значения.

РЕШЕНИЕ

Скетч в листинге 2.3 демонстрирует объявление переменных с плавающей запятой, иллюстрирует проблемы, которые могут возникнуть при сравнении чисел с плавающей запятой, а также показывает способы решения таких проблем.

Листинг 2.3. Операции с числами с плавающей запятой

```
/*
* Пример работы с числами с плавающей запятой
* Скетч инициализирует переменную с плавающей запятой value значением 1,1
* Затем значение этой переменной уменьшается в цикле на 0,1,
  пока она не станет равной 0
*/

float value = 1.1;

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

void loop()
{
   value = value - 0.1; // В каждой итерации цикла
                        // уменьшаем значение на 0,1
   if( value == 0)
   {
      Serial.println("Значение равно точно 0");
   }
```

[1] В англоязычной литературе используется термин «point» (точка) и соответствующий символ. В дальнейшем в тексте будет употребляться только термин «запятая».

```
  else if(almostEqual(value, 0))
  {
    Serial.print("Значение  ");
    Serial.print(value,7); // Выводим значение в монитор порта
                           // с точностью до 7 десятичных знаков
    Serial.println("  почти равно нулю, перезапускаем цикл декрементации");

    value = 1.1;
  }
  else
  {
    Serial.println(value);
  }
  delay(250);
}

// Возвращает true (истина) при небольшой разнице между a и b
bool almostEqual(float a, float b)
{
  const float DELTA = .00001; // Максимальная разница, чтобы значения
                              // считались почти равными
  if (a == 0) return fabs(b) <= DELTA;
  if (b == 0) return fabs(a) <= DELTA;
  return fabs((a - b) / max(fabs(a), fabs(b))) <= DELTA;
}
```

Обсуждение работы решения и возможных проблем

Операции над числами с плавающей запятой не дают абсолютно точных результатов, и возвращаемые значения могут содержать небольшие погрешности приближения. Эти ошибки возникают по той причине, что значения с плавающей запятой занимают громадный диапазон, в результате чего внутреннее представление такой величины может содержать только приблизительное значение. Поэтому при сравнении значений с плавающей запятой нужно проверять не точное равенство значений, а допустимую разность между ними.

Скетч выводит в окно монитора порта следующие данные:

```
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
```

```
Значение -0.0000001  почти равно нулю, перезапускаем цикл декрементации
1.00
0.90
```

Вывод начинается со значения 1,00 и продолжается с каждым последующим значением, меньшим на 0,1 предыдущего.

Логически рассуждая, после вычитания 0,1 из 0,1 должно выводиться сообщение: «Значение равно точно 0». Но значение результата такого вычитания никогда не будет равным точно 0 — оно может быть очень близким к нулю, но этого все равно недостаточно, чтобы успешно пройти тестирование:

```
if (value == 0)
```

Это объясняется тем, что хранение в памяти приближенного значения числа с плавающей запятой является единственным способом, эффективным в плане использования памяти, при которым числа с плавающей запятой могут содержать тот громадный диапазон значений, которые они могут представлять.

Эта проблема решается проверкой на близость значения переменной к желаемому значению, как и показано в скетче решения задачи (см. листинг 2.3).

Функция `almostEqual()` проверяет, находится ли значение переменной в заданных пределах целевого значения, и в таком случае возвращает значение `true`. Допустимые пределы задаются посредством константы `DELTA`, значение которой можно сделать большим или меньшим — как требуется. Функция `fabs()` (сокращение от floating-point absolute value, абсолютное значение числа с плавающей запятой) возвращает абсолютное значение переменной с плавающей запятой, которое используется в проверке разницы между указанными параметрами.

Прежде чем функция `almostEqual()` будет сравнивать разницу между значениями `a` и `b` со значением `DELTA`, она масштабирует эту разницу на максимальное значение или переменной `a`, или переменной `b`. Эта операция необходима, чтобы учесть тот факт, что точность значений с плавающей точкой зависит от их величины. Но в действительности, поскольку код сравнивает значение с 0, это выражение не является необходимым, поскольку когда или `a`, или `b` равно 0, то управление переходит к двум предшествующим строкам кода.

В табл. 2.3 демонстрируется, что бы происходило при разных порядках величины пар значений **Начальное** и **Целевое**. В столбце **Равные при** указывается конечная величина начального значения, при которой оба числа считаются равными. В столбце **Немасштабированная разница** указывается разница между значениями переменных `a` и `b`, когда функция `almostEqual()` определяет, что они почти равны друг другу. Наконец, в столбце **Масштабированная разница** указывается разница, используемая функцией `almostEqual()` для принятия своего решения. Как можно видеть, к тому времени, когда сравниваемые значения поднимаются до 100, немасштабированная разница превышает значение 0,00001 константы `DELTA`.

Переменные типа данных с плавающей запятой представляют числа только приближенно, поскольку в них задействуются лишь 32 бита для хранения всех возможных значений громаднейшего диапазона. При этом восемь из этих битов используются для десятичного множителя (экспоненты), оставляя 24 бита для знака и значения, что достаточно только для семи значащих десятичных разрядов.

Таблица 2.3. Сравнение чисел с плавающей запятой

Начальное	Целевое	Равные при	Немасштабированная разница	Масштабированная разница
11,1	10	9,9999962	0,0000038	0,0000004
101,1	100	100,0000153	0,0000153	0,0000002
1001,1	1000	1000,0002441	0,0002441	0,0000002

Хотя на 8-разрядных платах (например, платах Arduino Uno) переменные с плавающей запятой (`float`) и с двойной точностью (`double`) абсолютно одинаковые, последние поддерживают более высокую точность на 32-разрядных платах и на многих других платформах. Поэтому при импортировании кода из другой платформы, в котором используются переменные с плавающей запятой и с двойной точностью, проверьте, что для вашего приложения будет обеспечиваться достаточная точность.

Дополнительная информация

Более подробная информация по числам с плавающей запятой (типа `float`) приводится на странице справки веб-сайта Arduino (**https://oreil.ly/BHmBd**).

2.4. Работа с группами значений

ЗАДАЧА

Требуется создать и использовать группу значений (в языках программирования группы значений обычно называются *массивами*). Массивы могут быть одномерными — в виде простого списка, или с большей размерностью. Мы хотим узнать, как определить размер (количество элементов) массива и как затем обращаться к этим элементам.

РЕШЕНИЕ

Пример решения этой задачи приводится в листинге 2.4. Этот скетч создает два массива, один из которых содержит номера контактов, подключенных к кнопочным переключателям, а другой — номера контактов, подключенных к светодиодам. Соответствующая принципиальная схема подключения светодиодов и кнопочных переключателей к плате Arduino показана на рис. 2.1.

Листинг 2.4. Создание массивов и работа с ними

```
/*
Скетч array
Массив переключателей для управления массивом светодиодов
Дополнительная информация по переключателям приводится в главе 5
Дополнительная информация по светодиодам приводится в главе 7
*/
```

```
int inputPins[] = {2, 3, 4, 5}; // Создаем массив для хранения номеров
                                // контактов, подключенных к переключателям

int ledPins[] = {10, 11, 12, 13}; // Создаем массив для хранения номеров
                                  // контактов, подключенных к светодиодам

void setup()
{
   for (int index = 0; index < 4; index++)
   {
      pinMode(ledPins[index], OUTPUT);    // Задаем выходной режим
                                          // работы для контактов светодиодов
      pinMode(inputPins[index], INPUT_PULLUP); // Задаем входной режим
                                // работы для контактов переключателей
                                // и задействуем их встроенные повышающие резисторы
   }
}

void loop()
{
   for (int index = 0; index < 4; index++)
   {
      int val = digitalRead(inputPins[index]); // Считываем входное значение
      if (val == LOW) // Проверяем, нажата ли кнопка
      {
         digitalWrite(ledPins[index], HIGH); // Если нажата, включаем
                                             // соответствующий светодиод
      }
      else
      {
         digitalWrite(ledPins[index], LOW); // Если нет, выключаем светодиод
      }
   }
}
```

Если вы уже знакомы с входным режимом работы контактов ввода/вывода платы Arduino, возможно, вы привыкли заземлять через резистор контакт, на который подается положительный управляющий сигнал. Но когда для контакта задается входной режим работы посредством параметра `INPUT_PULLUP`, для контакта задействуется встроенный повышающий (подтягивающий) резистор, в результате чего внешний резистор (повышающий или понижающий) не требуется. Но следует обратить внимание на то, что в таком случае активным уровнем является низкий (`LOW`), а не высокий (`HIGH`).

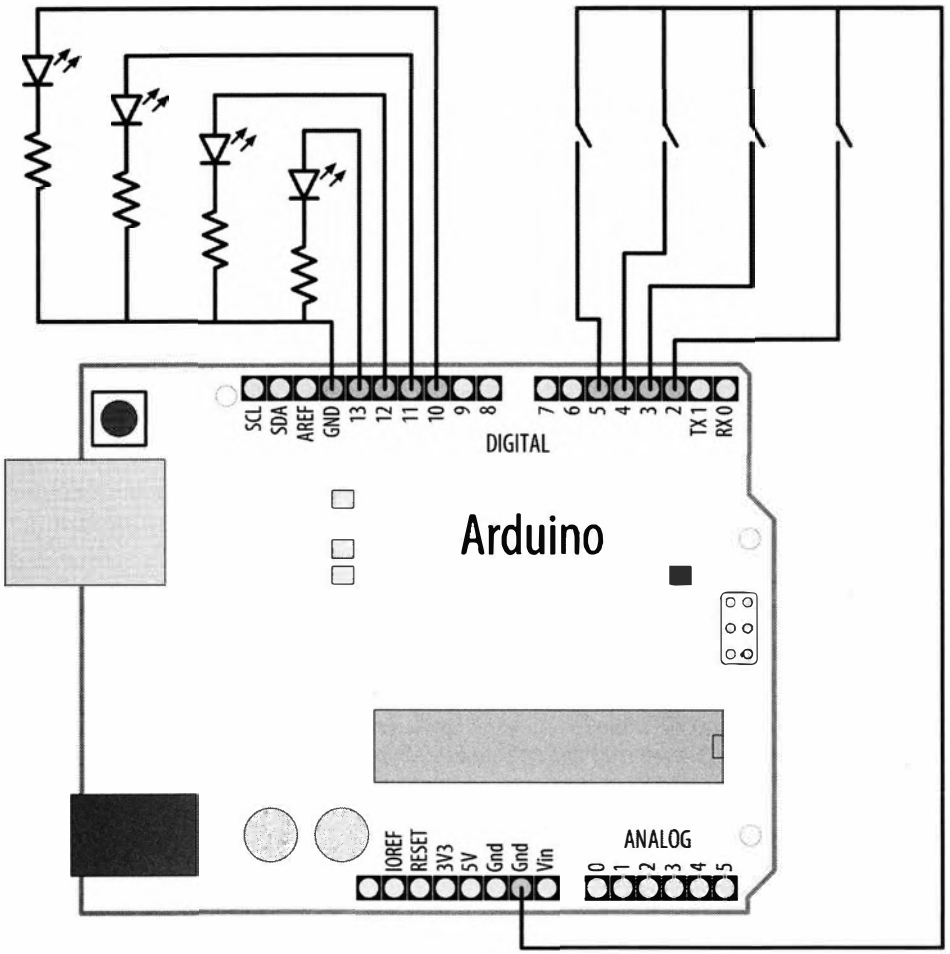


Рис. 2.1. Подключение светодиодов и переключателей для скетча из листинга 2.4

Обсуждение работы решения и возможных проблем

Массив представляет собой набор значений одинакового типа, хранящихся в последовательных ячейках памяти. Каждое значение массива называется *элементом*. Количество элементов определяет *размер массива* (не путать с *размерностью*).

В скетче решения (листинг 2.5) демонстрируется распространенное применение массивов в скетчах Arduino — хранение набора номеров контактов. В этом примере контакты подключены к кнопочным переключателям и светодиодам (светодиоды рассматриваются более подробно в *главе 5*). Важными частями этого примера являются код объявления массивов и обращения к их элементам.

Следующая строка кода *объявляет* (т. е. создает) массив целых чисел размером в четыре элемента и *инициализирует* (присваивает начальное значение) каждому элементу. Первому элементу присваивается значение 2, второму — 3 и т. д.:

```
int inputPins[] = {2,3,4,5};
```

Если не инициализировать элементы массива при его объявлении (например, когда значения будут доступны только при исполнении скетча), то значения каждого элемента нужно будет установить вручную. Объявление массива без инициализации его элементов осуществляется следующим образом:

```
int inputPins[4];
```

Этот оператор создает массив размером в четыре элемента. Если массив объявлен таким образом вне функции, каждому из его элементов присваивается начальное значение, равное 0. Если же массив объявлен таким образом внутри функции, например, `setup()` или `loop()`, то тогда его элементам будут присвоены произвольные значения. Число в квадратных скобках в операторе объявления массива задает его *размер*, т. е. количество элементов массива. В нашем случае массив состоит из четырех элементов и может содержать самое большее четыре целочисленных значения. При объявлении массива способом перечисления начальных значений его элементов (как это делается в примере) указывать размер массива нет необходимости, поскольку компилятор определяет его размер по количеству значений инициализации.

Поскольку язык Arduino основан на языках C/C++, в нем действуют концепции этих языков. В частности, в языках C/C++ элементам массивов, которые объявляются *глобально* (т. е. вне функций) указанием их размера, а не перечислением начальных значений их элементов, присваивается начальное значение 0. При объявлении массивов таким образом внутри функции значения их элементов будут неопределенными. Элементы, скорее всего, будут содержать значения, уже находящиеся в памяти, занимаемой элементами. Переменным, объявленным без инициализации (например, `int i;`), часто присваивается значение 0, но это не гарантируется, поэтому важно всегда инициализировать переменные, прежде чем использовать их значения.

Первый элемент массива обозначается как `имя_массива[0]`:

```
int firstElement = inputPins[0]; // Это первый элемент массива
inputPins[0] = 2; // Первому элементу массива присваивается значение 2
```

Указатель последнего элемента массива меньше размера массива на единицу, поэтому в нашем примере последним будет элемент 3:

```
int lastElement = inputPins[3]; // Это последний элемент массива
```

Может показаться странным, что четвертый элемент массива адресуется как `имя_массива[3]`, но поскольку первый элемент адресуется как `имя_массива[0]`, то это вполне логично, что и видно из следующего кода:

```
inputPins[0],inputPins[1],inputPins[2],inputPins[3]
```

В скетче примера обращение к элементам массива осуществляется с помощью цикла `for`:

```
for (int index = 0; index < 4; index++)
{
   pinMode(ledPins[index], OUTPUT);    // Задаем выходной режим
                                       // работы для контактов светодиодов
```

```
    pinMode(inputPins[index], INPUT_PULLUP); // Задаем входной режим
                              // работы для контактов переключателей
                              // и задействуем их встроенные повышающие резисторы
}
```

Этот цикл исполняет количество итераций, задаваемых переменной `index` с начальным значением 0 и конечным 3. Начинающие, да и не только начинающие программисты часто допускают ошибку случайного обращения к элементу массива, номер которого превышает количество элементов массива. Такая ошибка может проявляться разнообразными симптомами, и нужно уделять должное внимание, чтобы не допускать ее. Один из способов не допустить выхода за пределы размера массива — использовать константу, как показано в листинге 2.5.

Листинг 2.5. Использование константы, чтобы не допустить выхода за пределы массива

```
const int PIN_COUNT = 4;   // Определяем константу для количества элементов массива
int inputPins[PIN_COUNT] = {2,3,4,5};
int ledPins[PIN_COUNT] = {10, 11, 12, 13};

/* ... */

for(int index = 0; index < PIN_COUNT; index++)
{
   pinMode(ledPins[index], OUTPUT);
   pinMode(inputPins[index], INPUT_PULLUP);
}
```

Если код скетча содержит ошибку доступа к элементу массива вне его пределов, компилятор эту ошибку не обнаружит, но, скорее всего, при исполнении такого скетча произойдет фатальный сбой. Поэтому следует быть особо внимательным, чтобы осуществлять доступ только к элементам массива в пределах заданных границ. Использование константы для задания размера массива и обращения к его элементам в коде будет способствовать недопущению выхода за пределы массива.

Другим применением массивов является хранение в них строк текстовых символов. В языке Arduino они называются *символьными строками*, строчными значениями или просто строками. Символьная строка состоит из одного или больше символов и завершается непечатаемым символом `NULL` (значение 0).

Не следует путать символ `NULL` в конце символьной строки с символом 0. Значение кода ASCII для первого равно 0, а для второго — 48.

Способы работы со строками рассматриваются в *разд. 2.5* и *2.6*.

Дополнительная информация

Дополнительный материал по этим и подобным вопросам излагается в *разд. 5.2* и *7.1*.

2.5. Работа со строками в языке Arduino

ЗАДАЧА

Научиться манипулировать текстом: копировать его фрагменты, складывать отдельные фрагменты воедино и определять количество символов в текстовом фрагменте.

РЕШЕНИЕ

В предыдущем разделе упоминалось, что массивы можно использовать для хранения текстовых символов. Такие массивы текстовых символов обычно называются *строками*. Язык Arduino содержит объект строки `String`, предоставляющий разнообразные возможности для хранения и манипулирования текстом. Обратите внимание на то, что первая буква названия объекта `String` заглавная.

Слово `String` с первой заглавной буквой обозначает функциональность языка Arduino для работы с текстом, предоставляемую библиотекой String среды Arduino. А слово `string`, начинающееся со строчной буквы, означает группу символов, а не функциональность `String` среды Arduino[2].

В листинге 2.6 приводится код скетча, демонстрирующего работу с объектом `String` языка Arduino. Загрузите этот скетч в свою плату Arduino, запустите монитор порта и наблюдайте за выводимыми в нем результатами.

Листинг 2.6. Демонстрация работы со строками с помощью объекта `String`

```
/*
Скетч Basic_Strings
*/

String text1 = "Этот текст";
String text2 = " содержит больше символов";
String text3; // Значение присваивается кодом скетча

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Ожидаем открытия последовательного порта
                   // (плата Leonardo, другие 32-разрядные платы)

   Serial.print("text1 is ");              // text1 содержит
   Serial.print(text1.length());
   Serial.println(" characters long.");    // символов
```

[2] Это, скорее всего, относится только к английскому тексту, поскольку в русскоязычном слово «string» в этом контексте обычно переводится, как «строка». Хотя, возможно, существуют ситуации, когда это слово может использоваться и без перевода — например, как название переменной. Но важным моментом здесь является то, что название объекта `String` и библиотеки String не переводятся и пишутся с заглавной буквы.

```
    Serial.print("text2 is ");              // text2 содержит
    Serial.print(text2.length());
    Serial.println(" characters long.");    // символов

    text1.concat(text2);
    Serial.println("text1 now contains: "); // text1 теперь соджержит:
    Serial.println(text1);
}

void loop()
{
}
```

Зачем нужно условие `while(!Serial);`?

При запуске монитора порта в среде Arduino IDE выполняется сброс платы Arduino Uno и большинства 8-разрядных плат. Это означает, что короткое время после запуска монитора порта в его окно будет выведен весь текст, отправляемый на него по последовательному порту (в функции `setup()`). Но для плат Leonardo и плат с процессором SAMD запуск монитора порта не вызывает сброса этих плат. Это означает, что мы не сможем увидеть выводимый в монитор порта функцией `setup()` текст, поскольку не сможем открыть этот монитор достаточно быстро. По этой причине в нескольких функциях `setup()` в этой главе и в других главах книги используется условие `while(!Serial);`, которое приостанавливает дальнейшее исполнение кода скетча, пока не будет запущен монитор порта. Более подробная информация по этому обстоятельству приведена в *разд. «Особенности поведения аппаратных последовательных портов» главы 4*.

Обсуждение работы решения и возможных проблем

Приведенный в листинге 2.6 скетч создает три переменных типа `String`: `text1`, `text2` и `text3`. Переменные типа `String` обладают встроенными возможностями для манипулирования текстом. Функция `text1.length()` возвращает количество символов в строке, содержащейся в переменной `text1`. А функция `text1.concat(text2)` конкатенирует (добавляет) вторую строку к первой.

В результате в окне монитора порта отобразится следующий текст:

```
text1 is 9 characters long.
text2 is 20 characters long.
text1 now contains:
This text has more characters
```

Объединить две строки в одну можно также с помощью оператора сложения, как показано в следующем коде:

```
text3 = text1 + " and more";
Serial.println(text3);
```

Добавьте этот код в конец кода в функции `Setup()` и перезагрузите скетч в свою плату Arduino. Теперь в мониторе порта будет выводиться дополнительная строка текста:

```
This text has more characters and more
```

С помощью функции `indexOf()` можно найти положение первого вхождения определенного символа в строке, или с начала строки, или после определенного смещения. А функция `lastIndexOf()` позволяет найти положение вхождения определенного символа, начиная с конца строки. Как и в массивах, индексация строк в Arduino начинается с 0.

Вам, несомненно, придется повстречаться со скетчами Arduino, в которых вместо переменных типа `String` для представления строк используются массивы символов или указателей на символы. Дополнительная информация по использованию массивов символов для представления строк вместо переменных типа `String` приводится в *разд. 2.6*. А в *разд. 17.4* содержатся инструкции по хранению строковых констант во флеш-памяти вместо главной рабочей памяти RAM платы Arduino.

В коде наподобие следующего:

```
char oldString[] = "это массив символов";
```

строки хранятся в массивах символов, подобно тому, как это делается в языке C (см. *раз. 2.6*). А в следующем коде:

```
String newString = "это строчный объект";
```

для хранения строк используется переменная типа `String`. Массив символов можно преобразовать в переменную типа `String`, просто присвоив содержимое массива этой переменной:

```
char oldString[] = "Этот массив символов нужно поместить в переменную типа String";
String newString = oldString;
```

В табл. 2.4 приводится список и краткое описание других функций объекта `String` языка Arduino. Чтобы использовать эти функции, их нужно вызывать для существующего экземпляра объекта типа `String`, как показано в следующем примере:

```
int len = myString.length();
```

Таблица 2.4. *Функции объекта* `String` *языка Arduino*

Функция	Операция
`charAt(n)`	Возвращает *n*-й символ строки
`compareTo(S2)`	Сравнивает указанную строку, со строкой в параметре `S2`
`concat(S2)`	Возвращает новую строку, состоящую из указанной строки и строки в параметре `S2`
`endsWith(S2)`	Возвращает значение `true`, если указанная строка заканчивается подстрокой в параметре `S2`
`equals(S2)`	Возвращает `true`, если указанная строка в точности совпадает (с учетом регистра) со строкой в параметре `S2`
`equalsIgnoreCase(S2)`	То же самое, что и функция `equals()`, но без учета регистра
`getBytes(buffer,len)`	Копирует `len` количество символов указанной строки в указанный буфер

***Таблица 2.4** (окончание)*

Функция	Операция
`indexOf(S)`	Возвращает индекс вхождения строки (или одиночного символа) в указанную строку или –1 при отсутствии вхождения
`lastIndexOf(S)`	То же самое, что и `indexOf()`, но поиск ведется с конца указанной строки
`length()`	Возвращает количество символов в указанной строке
`remove(index)`	Удаляет символ, расположенный в указанной строке по указанному индексу
`remove(index, count)`	Удаляет указанное в `count` количество символов из указанной строки, начиная с указанного в `index` индекса
`replace(A,B)`	Заменяет все вхождения указанной в `A` строки (или символа) строкой, указанной в `B`
`reserve(count)`	Выделяет указанное в `count` количество байтов для обеспечения большей эффективности последующих строчных операций
`setCharAt(index,c)`	Сохраняет указанный в `c` символ по указанному в `index` индексу.
`startsWith(S2)`	Возвращает `true`, если указанная строка начинается с указанной в `S2` подстроки
`substring(index)`	Возвращает подстроку указанной строки, начинающуюся с указанного в `index` индекса и до конца строки
`substring(index,to)`	То же самое, что и `substring()`, но возвращаемая подстрока заканчивается в положении непосредственно перед положением, указанном в `to`
`toCharArray(buffer,len)`	Копирует `len` количество символов указанной строки в указанный в `buffer` буфер
`toFloat()`	Преобразовывает цифровые символы указанной строки в соответствующее значение с плавающей запятой
`toInt()`	Преобразовывает цифровые символы указанной строки в соответствующее целочисленное значение типа `int`
`toLowerCase()`	Преобразовывает все символы указанной строки в строчные
`toUpperCase()`	Преобразовывает все символы указанной строки в прописные
`trim()`	Удаляет из указанной строки все ведущие и замыкающие непечатаемые символы

Дополнительная информация по использованию и версиям этих функций приводится на соответствующих страницах справки веб-сайта Arduino.

Выбор между строками типа String и строками из массивов символов

Со встроенным типом данных `String` языка Arduino легче работать, чем с массивами символов, но эта легкость обеспечивается сложным кодом библиотеки String,

что предъявляет больше требований к плате Arduino и чревато большей вероятностью возникновения проблем.

Большая гибкость типа данных `String` возможна благодаря использованию динамического выделения памяти. Иными словами, при создании или модифицировании объекта `String` код скетча Arduino отправляет библиотеке C запрос на выделение новой области памяти, которую он должен освободить после завершения работы с этим объектом. Обычно такой процесс выделения и освобождения памяти проходит без каких бы то ни было осложнений, но 8-разрядные платы Arduino имеют настолько мало рабочей памяти RAM (2 Кбайт для плат Arduino Uno), что даже небольшие утечки памяти могут сильно повлиять на работу скетча. *Утечка памяти* происходит, когда вследствие несовершенства кода библиотеки или неправильного его применения выделенная память не освобождается после завершения ее использования. В результате объем доступной для работы скетча памяти будет медленно уменьшаться (пока не произойдет сброс платы). Родственной проблемой также является *фрагментация памяти*. При постоянном выделении и освобождении памяти непрерывные блоки свободной памяти будут становиться все меньшими, в результате чего попытка выделить память под новый объект `String` может закончиться неудачей, несмотря на наличие достаточного общего объема свободной памяти RAM.

Даже при отсутствии утечки памяти создание кода для проверки возможного сбоя запроса на создание объекта `String` по причине недостатка памяти будет сопряжено со сложностями. (Поведение функций объекта `String` языка Arduino похоже на поведение соответствующих функций языка Processing, но, в отличие от этой платформы, язык Arduino не поддерживает обработки исключений ошибок исполнения.) Ошибку нехватки динамической памяти очень трудно определить, поскольку скетч может исполняться без каких бы то ни было проблем в течение дней или даже недель, прежде чем начать барахлить вследствие нехватки памяти.

Использование массивов символов для работы со строками предоставляет разработчику возможность управления использованием памяти — при компиляции скетча выделяется фиксированный (статический) объем памяти, что предотвращает утечку памяти. В результате скетчу Arduino будет доступен неизменный объем памяти на протяжении всего времени его исполнения. Кроме этого, если по какой-либо причине и будет предпринята попытка выделить больший, чем доступный объем памяти, обнаружить такую ошибку гораздо легче благодаря наличию инструментов, позволяющих узнать объем выделенной статической памяти (см. обсуждение средства `avr-objdump` в *разд. 17.1*).

Но при работе с массивами символов легче совершить другую ошибку — при исполнении программы ничто не удержит ее от модифицирования памяти вне пределов объявленного массива. Например, ничто не сможет помешать нам выполнить присвоение: `myString[4] = 'A';` для массива символов, объявленного как `char myString[4];` (не забываем, что последним элементом этого массива является элемент `myString[3]`). Но кто знает, как используется область памяти, обозначаемая идентификатором `myString[4]`? И кто знает, не вызовет ли какую-либо проблему со-

хранение в этой области памяти символа 'A'? Скорее всего, такое сохранение вызовет сбой в работе скетча.

Таким образом, вследствие динамического выделения памяти использование объекта String создает риск появления нехватки памяти. А использование массивов символов требует от программиста внимания, чтобы не выйти за пределы объявленного массива. Поэтому библиотеку String следует использовать в тех случаях, когда требуется обрабатывать сложный текст и при этом не предполагается выполнять большой объем создания и модифицирования объектов String. А если строки будут создаваться и модифицироваться в бесконечном цикле, лучше использовать для их хранения массив символов большого размера и быть при этом особо осторожным в написании кода обработки такого массива, чтобы случайно не выйти за его пределы.

Предпочтение использованию массивов символов вместо объектов String также следует отдавать в больших скетчах, для которых требуется бóльшая часть доступной памяти RAM платы. Например, скетч примера Arduino StringToInt использует почти на 2 Кбайт больше флеш-памяти, чем если бы вместо объекта String для хранения строк использовался массив строк и преобразование в целое число выполнялось бы посредством функции atoi(). Для версии скетча, использующей объект String, кроме памяти для хранения собственно строки, также требуется немного дополнительной памяти RAM для хранения информации о выделения памяти под эту строку.

При наличии подозрений, что библиотека String или любая другая библиотека, которая использует динамически выделяемую память, может вызывать утечку памяти, в *разд. 17.2* рассматривается, как можно определить объем свободной памяти в любой момент времени. Проверьте объем свободной памяти RAM при запуске скетча и периодически проверяйте, не уменьшается ли он со временем. А при наличии подозрений на правильность работы библиотеки String, проверьте, нет ли вашей проблемы в списке известных проблем, выполнив поиск по ключевому слову String на странице соответствующего списка (**https://oreil.ly/Adyxc**).

Дополнительная информация

Среда разработки Arduino содержит примеры скетчей по работе с объектами String (меню **Файл | Примеры | 08.Strings**).

Дополнительная информация по объекту String и работе с ним приводится на странице справки этого объекта веб-сайта Arduino (**https://oreil.ly/GRq1L**).

На веб-сайте Arduino также имеется учебный материал по объекту String (**https://oreil.ly/XhtcM**).

2.6. Использование массива символов

ЗАДАЧА

Надо разобраться, как работать с «сырыми» строками: как создать строку, узнать количество символов в ней, как сравнивать, копировать и конкатенировать строки.

Базовый язык C не поддерживает возможности в стиле объекта `String` языка Arduino, поэтому нужно понимать код других платформ, предназначенный для работы с простыми массивами символов.

РЕШЕНИЕ

Массивы символов иногда называются *символьными строками*, или просто *строками*. Общее описание массивов языка Arduino приводится в *разд. 2.4*, где также дается список и краткое описание функций для работы со строками. Если у вас есть какой-либо опыт программирования на C/C++, вам должна быть знакома практика добавления в начале кода программы оператора:

```
#include <string.h>;
```

чтобы включить в свою программу функции для работы со строками, объявленные в этом файле. Среда Arduino IDE предоставляет функции для работы со строками без необходимости подключения этого заголовочного файла (подключение выполняется прозрачно для пользователя).

Строчные переменные в языке Arduino объявляются, как показано в листинге 2.7.

Листинг 2.7. Объявление строчных переменных в языке Arduino

```
char stringA[8];     // Объявляем строку длиной до 7 символов,
                     // плюс конечный символ NULL
char stringB[8] = "Arduino";  // Как и предыдущая строка кода,
                // но инициализируем строку значением Arduino
char stringC[16] = "Arduino"; // Как и предыдущая строка кода,
                  // но в строке еще осталось свободное место
char stringD[ ] = "Arduino";  // Компилятор выполняет инициализацию
                              // строки и вычисляет ее размер
```

Определить количество символов в строке без конечного символа `NULL` можно с помощью функции `strlen()`[3]:

```
int length = strlen(string); // Возвращает количество символов в указанной строке
```

Например, длина строки `stringA` в приведенном в листинге 2.7 примере объявления строк будет 0, а всех остальных — 7. Конечный символ строки `NULL` не включается функцией в количество символов строки.

Для копирования значения одной строчной переменной в другую используется функция `strcpy()`[4]:

```
strcpy(destination, source); // Копируем строки из переменной источника
                             // в переменную назначения
```

[3] Сокращение от *англ.* string length — длина строки.

[4] Сокращение от *англ.* string copy — копировать строку.

Функция `strncpy()` также копирует значение одной строчной переменной в другую, но только указанное количество символов строки источника. Эта возможность полезна, чтобы не допустить записывания с переменную назначения большего количества символов, чем она может вместить:

```
strncpy(назначение, источник, 6); // Копируем вплоть до 6 символов
                                  // из исходной строки в переменную назначения
```

Применение этой функции на практике демонстрируется в скетче, приведенном в листинге 2.9.

Для конкатенирования значения одной строчной переменной с другой используется функция `strcat()`[5]:

```
strcat(назначение, источник); // Добавляем строку источника в конец строки назначения
```

При копировании и конкатенировании строк всегда проверяйте, что в строке назначения есть достаточно места. Также не забывайте учитывать символ `NULL`, обозначающий конец строки.

Для сравнения значения одной строчной переменной с другой используется функция `strcmp()`[6]:

```
if(strcmp(str, "Arduino") == 0)
{
   // Если значение переменной str равно строке Arduino,
   // выполняем определенную операцию
}
```

Применение этой функции на практике демонстрируется в примерах *разд. 2.18*.

Обсуждение работы решения и возможных проблем

В программах на языке Arduino для представления текста используются массивы символов, называющиеся *строками*. Строка состоит из определенного количества символов и завершается символом `NULL` (код ASCII = 0). Символ `NULL` не отображается при выводе строки, но используется для обозначения конца строки.

Дополнительная информация

Только что рассмотренные функции `strxxx()` входят в состав библиотеки `string.h` языка C. Дополнительную информацию по этим функциям можно получить на многих интернет-страницах, посвященных языкам C/C++, — например, на справочных веб-сайтах cplusplus (**https://oreil.ly/oYYXf**) или CPP Reference (**https://oreil.ly/T0HJ2**).

---

[5] Сокращение от *англ.* string concatenate — конкатенировать строки.

[6] Сокращение от *англ.* string compare — сравнить строки.

2.7. Разбиение на группы текста, разделенного запятыми

ЗАДАЧА

Имеем строку, содержащую несколько элементов данных, разделенных запятой (или каким-либо иным разделяющим символом). Требуется разбить строку по этим разделителям на составляющие подстроки.

РЕШЕНИЕ

Эта задача решается с помощью скетча, код которого приводится в листинге 2.8.

Листинг 2.8. Разбиение строки по разделителям на подстроки

```
/*
* Скетч SplitSplit
* Разбивает строку по запятым на подстроки
*/

String text = "Peter,Paul,Mary";   // Строка примера
String message = text;             // Переменная для исходной строки
int commaPosition;                 // Переменная для положения следующей
                                   // запятой в строке

void setup()
{
   Serial.begin(9600);
   while(!Serial);          // Ожидаем открытия последовательного порта
                            // (плата Leonardo, другие 32-разрядные платы)
   Serial.println(message); // Выводим в окно монитора порта исходную строку
   do
   {
      commaPosition = message.indexOf(',');
      if(commaPosition != -1)
      {
         Serial.println( message.substring(0,commaPosition));
         message = message.substring(commaPosition+1, message.length());
      }
      else
      { // Переходим сюда после обнаружения последней запятой
         if(message.length() > 0)
         Serial.println(message); // Если после последней запятой
                                  // есть текст, выводим его в окно монитора порта
      }
   }
   while(commaPosition >=0);
}
```

```
void loop()
{
}
```

В окне монитора порта отобразится следующий текст:

```
Peter,Paul,Mary
Peter
Paul
Mary
```

Обсуждение работы решения и возможных проблем

Для извлечения из исходной строки расположенных между запятыми подстрок в скетче используются функции объекта `String`. Следующая строка кода:

```
commaPosition = message.indexOf(',');
```

присваивает переменной `commaPosition` значение позиции первой запятой в переменной объекта `String`, называющейся `message` (в случае, если в исходной строке не обнаружится ни одной запятой, этой переменной присваивается значение –1). При обнаружении запятой функция `substring` выводит в окно монитора порта текст исходной строки, начиная с первого символа до найденной запятой, но не саму запятую. Затем этот текст и завершающая его запятая удаляются из исходной строки следующей строкой кода:

```
message = message.substring(commaPosition+1, message.length());
```

Здесь функция `substring` возвращает подстроку исходной строки, начиная с позиции `commaPosition+1` (позиция сразу же после первой найденной запятой) и до конца исходной строки. В результате переменная `message` будет содержать только подстроку исходной строки, следующую за первой запятой. Этот процесс повторяется до тех пор, пока в исходной строке больше не останется запятых (т. е. значение переменной `commaPosition` станет равно –1).

Опытные программисты могут решить эту задачу, используя низкоуровневые функции стандартной библиотеки языка C. Скетч в листинге 2.9 реализует ту же самую функциональность, что и скетч в листинге 2.8, но посредством низкоуровневых функций языка C.

Листинг 2.9. Разбиение на подстроки с помощью низкоуровневых функций

```
/*
* Скетч strtok
* Разбивает строку по запятым на подстроки
*/

const int MAX_STRING_LEN = 20; // Присваиваем этой переменной значение
       // максимального количества символов возможной исходной строки
```

```
char stringList[] = "Peter,Paul,Mary"; // Строка примера

char stringBuffer[MAX_STRING_LEN+1];   // Статический буфер для вычислений и вывода

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Ожидаем открытия последовательного порта
                   // (плата Leonardo, другие 32-разрядные платы)

   char *str;
   char *p;
   strncpy(stringBuffer, stringList, MAX_STRING_LEN); // Копируем исходную строку
   Serial.println(stringBuffer);                      // Выводим в окно
                                                      // монитора порта исходную строку

   for( str = strtok_r(stringBuffer, ",", &p); // Разбиваем исходную строку по запятым,
        str;                                   // пока значение str
                                               // не станет равно NULL
        str = strtok_r(NULL, ",", &p)  // Извлекаем последующие подстроки
      )
   {
      Serial.println(str);
   }
}

void loop()
{
}
```

Хотя в языке Arduino разрешается использовать указатели, эта практика не приветствуется, поскольку усложняет понимание кода для начинающих программистов. На практике в скетчах примеров редко используются указатели или другие продвинутые возможности языка C. Дополнительная информация по рекомендуемому стилю программирования на языке Arduino предоставляется на странице Arduino Style Guide (https://oreil.ly/a5mv9) веб-сайта Arduino.

Базовую функциональность скетча реализует функция `strtok_r()` (это версия функции `strtok()`, поддерживаемая компилятором Ardino). При первом вызове этой функции ей передается строка, которую нужно разбить на подстроки по запятым. Но при каждом обнаружении новой подстроки функция `strtok_r()` перезаписывает все символы исходной строки, поэтому ей следует передавать копию этой строки, как и показано в примере скетча. В каждом последующем вызове этой функции символ `NULL` указывает ей, что нужно переходить к следующей подстроке. В приведенном примере каждая обнаруженная подстрока выводится в окно монитора порта. Для отслеживания ее местонахождения в исходной строке функция `strtok_r()` использует переменную с оператором получения адреса `*p`, которая передается ей с оператором ссылки как `&p`.

Процедура разбиения на части строки, содержащей только числа, приведена в *разд. 4.5*. В частности, там рассматривается извлечение из последовательного потока данных числовых значений, разделенных запятыми.

Дополнительная информация

Дополнительная информация по функциям `strtok_r()` и `strcmp()` языка C для работы со строками приводится на странице **https://oreil.ly/-Zehk** веб-сайта AVR Libc.

В *разд. 2.5* рассматривается работа со строками с помощью функций объекта `String` библиотеки Arduino.

Дополнительная информация по функциям `strtok_r()` и `strcmp()` также приводится на странице **https://oreil.ly/sKSld** веб-сайта **Man7.org**.

2.8. Преобразование числа в строку

ЗАДАЧА

Требуется преобразовать числовое значение в соответствующую строку — например, для вывода на жидкокристаллический дисплей (ЖКД) или какое-либо другое устройство.

РЕШЕНИЕ

Преобразовать число в текстовое представление можно, присвоив это число в качестве значения переменной типа `String`. Для присвоения можно использовать как собственно число, так и содержимое числовой переменной. Например, можно использовать такой код:

```
String myNumber = String(1234);
```

Или такой:

```
int value = 127;
String myReadout = "The reading was ";
myReadout.concat(value);
```

А также такой:

```
int value = 127;
String myReadout = "The reading was ";
myReadout += value;
```

Обсуждение работы решения и возможных проблем

Самый простой способ преобразовать число в текст для отображения на ЖКД или другом дисплее с последовательным интерфейсом — использовать встроенную в ЖКД возможность этого преобразования и библиотеку Serial (см. *разд. 4.2*). Но если устройство отображения не оснащено такой встроенной возможностью

(см. *главу 13*) или же в скетче требуется манипулировать числом как текстом, тогда нужно искать другие подходы.

Одним из таких подходов будет использование класса `String` языка Arduino — при присвоении числового значения переменной типа `String` оно автоматически преобразовывается в текст. Числовое значение можно добавить в конец текстовой строки, используя функцию `concat()` или оператор сложения (+).

Оператор сложения (+) можно использовать как с числами, так и со строками, но это дает разные результаты в каждом случае.

В результате исполнения следующего кода переменной `number` присваивается целочисленное значение 13:

```
int number = 12;
number += 1;
```

Когда же используется переменная типа `String`, результатом будет текстовая строка `"121"`:

```
String textNumber = "12";
textNumber += 1;
```

До введения класса `String` в коде скетчей Arduino часто использовались функции `itoa()` и `ltoa()`. Названия функций созданы из сокращений описаний их назначения: **i**nteger **to** ASCII (целое число в ASCII) и **l**ong **to** ASCII (длинное целое число в ASCII) для `itoa()` и `ltoa()` соответственно. Подход с использованием класса `String` более легкий, но в случае предпочтения или необходимости работы с массивами символов (см. *разд. 2.6*) для преобразования числовых значений в текст можно использовать и эти функции.

Функциям `itoa()` и `ltoa()` передаются три параметра: числовое значение, подлежащее преобразованию в текст, буфер для хранения строки результата и основание системы счисления (10 — для десятичного числа, 16 — для шестнадцатеричного и 2 — для двоичного).

Скетч в листинге 2.10 иллюстрирует процесс преобразования числовых значений в текст с использованием функции `ltoa()`.

Листинг 2.10. Преобразование числа в текст с помощью функции `ltoa()`

```
/*
* Скетч NumberToString
* Преобразовывает число в текст
*/

char buffer[12]; // Для типа данных long требуется буфер размером
                 // 12 символов (включая знак "минус" и конечный символ NULL)
```

```
void setup()
{
   Serial.begin(9600);
   while(!Serial);

   long value = 12345;
   ltoa(value, buffer, 10);

   Serial.print( value);
   Serial.print(" has ");
   Serial.print(strlen(buffer));
   Serial.println(" digits");

   value = 123456789;
   ltoa(value, buffer, 10);

   Serial.print( value);
   Serial.print(" has ");
   Serial.print(strlen(buffer));
   Serial.println(" digits");
}

void loop()
{
}
```

Размер буфера должен быть достаточно большим, чтобы в нем можно было сохранить максимальное количество символов, из которых может состоять конечная строка. Для 16-разрядных десятичных целых чисел потребуется буфер размером в семь символов (пять цифр, возможный знак минуса и конечный символ `NULL`), а для 32-разрядных длинных (`long`) целых чисел требуется уже 12 символов (10 цифр, знак минуса и конечный символ `NULL`). При этом компилятор не предупреждает о превышении размера буфера. Ошибка, связанная с неправильным назначением размера буфера, может вызывать непредсказуемые последствия, поскольку запись за пределами буфера исказит какую-либо часть памяти, которая может использоваться скетчем. Самый простой способ избежать такой ошибки — всегда задавать буфер размером в 12 символов и всегда использовать функцию `ltoa()`, поскольку она работает как с 16-разрядными, так и с 32-разрядными значениями.

2.9. Преобразование текста в число

ЗАДАЧА

Требуется преобразовать текстовое представление числа в соответствующее числовое значение. Например, полученное по последовательному каналу связи текстовое значение числа нужно использовать в вычислениях как целое число или число с плавающей запятой.

РЕШЕНИЕ

Эту задачу можно решить несколькими способами. Строку, полученную из последовательного потока данных, можно преобразовать в число с помощью функции `parseInt()`. Примеры реализации такого преобразования приводятся в *разд. «Обсуждение работы решения и возможных проблем»* этого раздела, а также в *разд. 4.3*.

Другой подход к преобразованию текстовых строк, представляющих числа, реализуется использованием функции языка C `atoi()` (для переменных типа `int`) или `atol()` (для переменных типа `long`).

Этот подход демонстрируется в скетче, приведенном в листинге 2.11. Здесь конец строки цифровых символов указывается нецифровым символом (или заполнением буфера). Загрузите скетч в свою плату Arduino, запустите монитор порта, введите несколько цифровых символов в строку ввода, завершив их каким-либо нецифровым символом, и нажмите кнопку **Отправить** или клавишу <Enter>. Чтобы не заморачиваться с вводом нецифрового символа, можно просто установить опцию **NL (Новая строка)** в первом выпадающем списке в нижней правой части монитора порта.

Листинг 2.11. Преобразование текста в число посредством функции `atoi()`

```
/*
* Скетч StringToNumber
* Преобразовывает строку цифровых символов в соответствующее число
*/

int blinkDelay;     // Переменная для хранения значения
                    // частоты мигания светодиода
char strValue[6];   // Переменная для хранения исходной строки.
                    // Размер массива должен быть достаточным для хранения
                    // всех символов строки и ее конечного символа NULL
int index = 0;      // Указатель для массива, хранящего полученные цифровые символы

void setup()
{
   Serial.begin(9600);
   pinMode(LED_BUILTIN, OUTPUT); // Задаем режим вывода для контакта
                                 // встроенного светодиода
}

void loop()
{
   if( Serial.available())
   {
      char ch = Serial.read();
      if(index < 5 && isDigit(ch))
```

```
      {
         strValue[index++] = ch; // Добавляем в строку символ ASCII
      }
      else
      {
         // Идем сюда, когда буфер заполнен или получен
         // первый нецифровой символ
         strValue[index] = 0;     // Завершаем строку символом NULL
         blinkDelay = atoi(strValue); // Используем функцию atoi()
         // для преобразования строки цифровых символов
         // в соответствующее числовое значение типа int
         index = 0;
      }
   }
   blink();
}

void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
   delay(blinkDelay/2);      // Пауза в половину заданного периода мигания
   digitalWrite(LED_BUILTIN, LOW);
   delay(blinkDelay/2);      // Пауза в половину заданного периода мигания
}
```

Обсуждение работы решения и возможных проблем

Функция `atoi()` (**ASCII to int** — ASCII в целое число) преобразовывает строку цифровых символов в целое число типа `int`, а функция `atol()` (**ASCII to long**) — в целое число типа `long`. Прежде чем вызывать эти функции преобразования, необходимо принять всю строку и сохранить ее в массиве символов. В коде создается массив символов `strValue[6]` для хранения до пяти цифровых символов и конечного символа `NULL`. Массив заполняется цифровыми символами, получаемыми функцией `Serial.read()`, пока она не получит первый нецифровой символ. В конец строки полученных цифровых символов добавляется символ `NULL`, после чего вызывается функция `atoi()`, которая преобразовывает хранящиеся в массиве цифровые символы в соответствующее числовое значение. Полученное значение затем присваивается переменной `blinkDelay()`.

После чего вызывается функция `blink()`, которая мигает встроенным светодиодом с частотой, определяемой значением переменной `blinkDelay`.

Как упоминалось в *разд. 2.4*, необходимо быть внимательным, чтобы не выйти за пределы массива. Подробная информация о том, как это делать, приводится там в *разд. «Обсуждение работы решения и возможных проблем»*.

Язык Arduino также содержит функцию `parseInt()`, посредством которой можно извлекать целые числа из данных, получаемых по последовательному (объект

Serial) или сетевому (объект Ethernet) интерфейсу (или по любому интерфейсу, реализуемому объектом, производным от класса Stream). В листинге 2.12 приводится фрагмент кода, который использует эту функцию для преобразования последовательностей цифровых символов в соответствующие числовые значения типа int. Этот подход подобен способу преобразования, к которому мы прибегали в скетче, приведенном в листинге 2.11, но здесь не используется буфер и, соответственно, количество цифровых символов не ограничено пятью символами.

Листинг 2.12. Преобразование строки цифровых символов в число посредством функции Serial.parseInt()

```
void loop()
{
   if( Serial.available())
   {
      int newValue = Serial.parseInt();
      if (newValue != 0)
      {
         blinkDelay = newValue;
         Serial.print("New delay: ");
         Serial.println(blinkDelay);
      }
   }
blink();
}
```

Методы парсинга потоковых данных — такие как функция parseInt(), используют тайм-аут для возврата управления скетчу, если в течение заданного интервала не прибывает никаких данных. По умолчанию время тайм-аута равно одной секунде, но его можно изменить, вызвав функцию setTimeout():

```
Serial.setTimeout(1000 * 60); // ожидаем данные максимум одну минуту
```

Если по истечении времени тайм-аута не был получен символ завершения последовательности цифровых символов, функция parseInt() (и другие функции класса Stream для парсинга потоковых данных) возвратит всю последовательность, полученную на момент истечения времени тайм-аута. Если полученная последовательность не содержит цифровых символов, возвращается значение ноль.

Дополнительная информация

Дополнительная информация по функции atoi()приводится на веб-сайте AVR Libc (**https://oreil.ly/JOGu-**).

Дополнительную информацию по рассмотренным здесь низкоуровневым функциям можно получить на многих интернет-страницах, посвященных языкам C/C++, — например, на справочных веб-сайтах cplusplus (**https://oreil.ly/zzJ2O**) или CPP Reference (**https://oreil.ly/Gwo1a**). Дополнительная информация по использованию функции parseInt() с объектом Serial приводится в *разд. 4.3* и *4.5*.

2.10. Организация кода в функциональные блоки

ЗАДАЧА

Мы хотим узнать, как добавлять в скетч функции, а также научиться планировать общую структуру скетча.

РЕШЕНИЕ

Для организации выполняемых скетчем действий в функциональные блоки используются функции. Они позволяют оформить определенную функциональность в четко определенные вводы (предоставляемую функции информацию) и выводы (выдаваемую функцией информацию), что упрощает задачу структурирования, обслуживания и повторного использования кода. Мы уже знакомы с двумя функциями, которые содержит каждый скетч Arduino: `setup()` и `loop()`. При создании функции объявляется тип возвращаемого ею значения, ее название и любые параметры (значения), передаваемые функции при ее вызове.

Термины «функция» и «метод» обозначают четко определенные блоки кода, которые можно вызывать одним оператором в других частях программы. Термин «функция» используется в языке C. А в объектно-ориентированных языках — таких как C++, которые предоставляют функциональность посредством классов, используется термин «метод». Язык Arduino использует смесь этих стилей: в примерах скетчей в целом реализуются функции в стиле языка C, а библиотеки в основном пишутся для предоставления методов классов языка C++. В этой книге мы используем термин «функция», если только код не предоставляется посредством класса. Однако не расстраивайтесь, если вы не можете понять разницу между двумя этими терминами, — просто считайте их равнозначными.

В листинге 2.13 приводится код для простой функции, которая просто мигает светодиодом. Ей не передается никаких параметров, и она не возвращает никаких значений. Последний факт обозначен использованием ключевого слова `void` в объявлении функции.

Листинг 2.13. Функция мигания светодиодом

```
// Мигает светодиодом один раз
void blink1()
{
   digitalWrite(LED_BUILTIN, HIGH); // Включаем светодиод
   delay(500);                      // Пауза длительностью в 500 миллисекунд
   digitalWrite(LED_BUILTIN, LOW);  // Выключаем светодиод
   delay(500);                      // Пауза длительностью в 500 миллисекунд
}
```

А функции в листинге 2.14 передается параметр `count` типа `int`, определяющий количество миганий светодиода.

Листинг 2.14. Функция мигания светодиодом с параметром

```
// Мигает светодиодом количество раз, указанных в параметре count
void blink2(int count)
{
   while(count > 0 ) // Итерируем цикл, пока значение count
                     // не станет равным нулю
   {
      digitalWrite(LED_BUILTIN, HIGH); // Включаем светодиод
      delay(500);                      // Пауза длительностью в 500 миллисекунд
      digitalWrite(LED_BUILTIN, LOW);  // Выключаем светодиод
      delay(500);                      // Пауза длительностью в 500 миллисекунд
      count = count -1;                // Декрементируем значение count
   }
}
```

Опытные программисты могут отметить, что обе функции можно назвать одинаково: `blink`, поскольку компилятор различит их по типу значений параметра. Это поведение называется *перегрузкой функции*. Функция `Serial.print()`, рассматриваемая в *разд. 4.2*, является распространенным примером перегрузки функции. Другой пример перегрузки функции приводится в *разд. «Обсуждение работы решения и возможных проблем» разд. 4.6*.

Эта версии функции проверяет, равно ли нулю значение переменной `count`. Если нет, то выполняется мигание светодиодом и значение переменной `count` уменьшается на единицу. Процесс повторяется до тех пор, пока значение `count` не станет равным нулю.

В некоторых пособиях *параметр* функции иногда называется *аргументом*. Для практических целей эти термины можно считать равнозначными.

В листинге 2.15 приводится пример скетча, содержащий функцию, которая принимает значение в параметре и возвращает вычисленное ею значение. Значение, передаваемое функции в параметре, определяет длительность в миллисекундах включенного и выключенного состояний светодиода. Функция мигает светодиодом, пока не будет нажата кнопка, после чего она возвращает значение количества раз мигания светодиодом. Схема подключения светодиода и кнопки такая же, как и для скетча в *разд. 5.2* (см. рис. 5.5).

Листинг 2.15. Пример функции, принимающей параметр и возвращающей значение

```
/*
Скетч blink3
Демонстрирует вызов функции, принимающей значение в параметре
и возвращающей вычисленное ею значение.
Светодиод начинает мигать при запуске скетча и перестает мигать
при нажатии кнопки, подключенной к контакту 2.
```

```
В окно монитора порта выводится значение количества миганий светодиода.
*/

const int inputPin = 2; // Контакт для подключения кнопки

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(inputPin, INPUT);
   digitalWrite(inputPin,HIGH); // Подключаем внутренний
                                // повышающий резистор (см. разд. 5.2)
   Serial.begin(9600);
}

void loop()
{
   Serial.println("Press and hold the switch to stop blinking");
                // Чтобы прекратить мигание,
                // нажмите и удерживайте кнопку
   int count = blink3(250); // Мигаем светодиодом: 250 мс включен
                            // и 250 мс выключен
   Serial.print("The number of times the switch blinked was ");
               //Количество миганий светодиода равно:
   Serial.println(count);
   while(digitalRead(inputPin) == LOW)
   {
      // Ничего не делаем, пока не будет отпущена кнопка
   }
}
// Мигает светодиодом, используя значение задержки, указанное в параметре period
// Возвращает количество миганий светодиода
int blink3(int period)
{
   int blinkCount = 0;
   while(digitalRead(inputPin) == HIGH) // Мигаем, пока не будет нажата кнопка
   // (При нажатии кнопки на контакте установится низкий уровень)
   {
      digitalWrite(LED_BUILTIN, HIGH);
      delay(period);
      digitalWrite(LED_BUILTIN, LOW);
      delay(period);
      blinkCount = blinkCount + 1; // Инкрементируем счетчик
   }
   // Переходим сюда, когда на контакте inputPin больше нет
   // высокого уровня (Что означает нажатие кнопки)
   return blinkCount; // Возвращаем количество миганий светодиода
}
```

Функция объявляется как *прототип* — описание ее названия, типов значений, передаваемых функции, и типа возвращаемого ею значения. Процесс сборки языка Arduino делает объявление функции прозрачным для пользователя, что делает ненужным следование стандартному требованию языка C объявлять функцию явно.

Обсуждение работы решения и возможных проблем

Код, приведенный в этом решении, иллюстрирует три формы функции, с которыми вам придется встретиться. Функция `blink1` не имеет ни параметров, ни возвращаемого значения. Она имеет следующую конструкцию:

```
void blink1()
{
   // Сюда идет код реализации действия функции
}
```

Функция `blink2` принимает один параметр, но не возвращает значение:

```
void blink2(int count)
{
   // Сюда идет код реализации действия функции
}
```

Функция blink3 принимает один параметр и возвращает значение:

```
int blink3(int period)
{
   int result = 0;
   // Сюда идет код реализации действия функции
   return result; // Возвращаемое значение
}
```

Указание типа данных перед названием функции обозначает тип возвращаемого ею значения. Если функция не возвращает никакого значения, вместо типа данных указывается ключевое слово `void`. При *определении функции* (т. е. написании кода, определяющего функцию и ее действие), после закрывающей фигурной скобки точка с запятой на ставится. Но оператор вызова функции должен завершаться точкой с запятой.

Большинство функций, с которыми вам придется встретиться, будут той или иной вариацией одной из этих форм.

Идентификатор типа данных в начале определения функции сообщает компилятору (и напоминает разработчику), какой тип данных возвращает функция. Для функций `blink1` и `blink2` идентификатор `void` означает, что эти функции не возвращают никакого значения. А идентификатор `int` для функции `blink3` означает, что она возвращает числовое значение типа `int`. При создании функций следует выбирать тип данных возвращаемого значения, соответствующий выполняемому функцией действию.

Рекомендуется присваивать функциям значащие названия. Кроме этого, распространенной практикой именования функций является создание их названий из нескольких слов, пишущихся без пробелов, — при этом все слова, кроме первого, пишутся с заглавной буквы. Никто не запрещает использовать тот стиль именования функций, который является удобным для вас лично, но последовательность в стиле именования помогает в понимании вашего кода другими.

Функция `blink2` имеет параметр `count`, через который функции передается значение при ее вызове. Функция `blink3` также имеет параметр, но с другим названием — `period`.

Тело функции (т. е. код внутри фигурных скобок) выполняет требуемое действие, которое для функции `blink1` состоит в мигании светодиодом, т. е. периодическим его включением и выключением. Код функции `blink2` итерирует в цикле `while` количество раз, указанное в параметре `count`, мигая светодиодом при каждой итерации. А функция `blink3` мигает светодиодом до тех пор, пока не будет нажата кнопка, после чего она возвращает вызывающему коду значение — количество раз миганий светодиода до нажатия кнопки.

Дополнительная информация

Дополнительная информация по организации и работе с функциями Arduino приводится на веб-сайте Arduino (**https://oreil.ly/Ww2Hw**).

2.11. Возвращение функцией нескольких значений

ЗАДАЧА

Требуется, чтобы функция возвращала не одно значение, а два или больше. В *разд. 2.10* были приведены примеры наиболее распространенного вида функции, которая возвращает только одно значение или вообще не возвращает значения. Но иногда требуется, чтобы функция возвратила больше чем одно значение.

РЕШЕНИЕ

Эту задачу можно решить несколькими способами. Проще всего изменить в функции соответствующие глобальные переменные. Пример такой функции приводится в листинге 2.16.

Листинг 2.16. Функция, модифицирующая глобальные переменные

```
/*
Скетч swap
Демонстрирует модификацию двух значений с использованием глобальных переменных
*/

int x; // Переменные x и y — глобальные
int y;
```

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

void loop()
{
   x = random(10); // Создаем пару произвольных чисел
   y = random(10);

   Serial.print("The value of x and y before swapping are: ");
     // Переменные x и y до обмена имеют значения:
   Serial.print(x); Serial.print(","); Serial.println(y);
   swap();

   Serial.print("The value of x and y after swapping are: ");
    // Переменные x и y после обмена имеют значения:
   Serial.print(x); Serial.print(","); Serial.println(y);Serial.println();

   delay(1000);
}

// Меняем местами значения двух глобальных переменных
void swap()
{
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

Функция `swap` модифицирует два значения, используя глобальные переменные. *Глобальные переменные* представляют собой такие переменные, чьи значения доступны из любого места программы любому коду. Несмотря на простоту концепции глобальных переменных, опытные программисты избегают использовать их, поскольку очень легко непреднамеренно изменить значение или тип такой переменной вне функции где-либо в программе, в результате чего функция не будет работать должным образом.

Более безопасное и элегантное решение заключается в передаче функции ссылки на значения, которые нужно изменить, и разрешении функции использовать эти ссылки, чтобы модифицировать значения. В листинге 2.17 приводится пример реализации такого подхода.

Листинг 2.17. Возврат функцией двух значений с использованием ссылки

```
/*
Скетч functionReferences
Демонстрирует возврат двух значений посредством передачи ссылок
*/
```

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

void loop()
{
   int x = random(10); // Создаем пару произвольных чисел
   int y = random(10);

   Serial.print("The value of x and y before swapping are: ");
         // Переменные x и y до обмена имеют значения:
   Serial.print(x); Serial.print(","); Serial.println(y);
   swapRef(x,y);
   Serial.print("The value of x and y after swapping are: ");
        // Переменные x и y после обмена имеют значения:
   Serial.print(x); Serial.print(","); Serial.println(y);Serial.println();
   delay(1000);
}

// Меняем местами значения двух переменных
void swapRef(int &value1, int &value2)
{
   int temp;
   temp = value1;
   value1 = value2;
   value2 = temp;
}
```

Наконец, в качестве еще одного подхода можно использовать *структуру* C, которая может содержать несколько *полей*, позволяя передавать и возвращать какие угодно виды данных. Этот подход демонстрируется в листинге 2.18.

Листинг 2.18. Передача и возврат данных с помощью структуры

```
/*
Скетч struct
Демонстрирует возвращение функцией более одного значения с использованием структуры
*/

struct Pair
{
   int a, b;
};

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

```
void loop()
{
   int x = random(10); // Создаем пару произвольных чисел
   int y = random(10);
   struct Pair mypair = {random(10), random(10)};
   Serial.print("The value of x and y before swapping are: ");
              // Переменные x и y до обмена имеют значения:
   Serial.print(mypair.a); Serial.print(","); Serial.println(mypair.b);
   mypair = swap(mypair);
   Serial.print("The value of x and y after swapping are: ");
                // Переменные x и y после обмена имеют значения:
   Serial.print(mypair.a); Serial.print(",");
   Serial.println(mypair.b);Serial.println();
   delay(1000);
}

// Меняем местами значения двух переменных
Pair swap(Pair pair)
{
   int temp;
   temp = pair.a;
   pair.a = pair.b;
   pair.b = temp;
   return pair;
}
```

Обсуждение работы решения и возможных проблем

Функция `swapRef()` похожа на функции с параметрами, рассмотренными в *разд. 2.10*, но символ амперсанда (`&`) в определении функции указывает на то, что передаваемые параметры являются ссылками на значения, а не самими значениями. Это означает, что изменение значений внутри функции также изменит значение переменной, указываемой при вызове функции. Чтобы понять, как работает эта функция, сначала исполните скетч в листинге 2.17 и убедитесь, что значения параметров меняются. Затем модифицируйте код скетча, удалив знаки амперсанда в определении функции.

Модифицированная строка кода должна выглядеть следующим образом:

```
void swapRef(int value1, int value2)
```

Исполнение модифицированного кода покажет, что теперь значения не меняются — изменения, выполненные внутри функции, являются локальными и теряются по возврату функции.

В функции `swapPair()` (см. листинг 2.18) используется возможность языка C, называющаяся `struct` (от structure, структура). Структура может содержать любое количество переменных простых типов или указателей. Для структуры резервируется объем памяти, равный размеру содержащихся в ней элементов. В нашем примере

структура `Pair` займет четыре байта на 8-разрядной плате Arduino и восемь — на 32-разрядной плате. Если вы знакомы с объектно-ориентированным программированием, у вас может возникнуть соблазн рассматривать структуру как нечто, похожее на класс, но структура не является ничем большим, чем содержащиеся в ней данные.

2.12. Условные операции

ЗАДАЧА

Требуется исполнить блок кода только при наличии определенного условия. Например, включить светодиод, если нажата кнопка или если величина аналогового значения превысит определенное пороговое значение.

РЕШЕНИЕ

Пример скетча для решения этой задачи приводится в листинге 2.19. Для скетча используется схема подключения светодиода и кнопки, показанная на рис. 5.5 в *разд. 5.2*.

Листинг 2.19. Выполнение действия при наличии определенного условия

```
/*
* Скетч Pushbutton
* Нажатие кнопки, подключенной к цифровому контакту 2 платы,
  включает встроенный светодиод
*/

const int inputPin = 2; // Контакт для подключения кнопки

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);     // Задаем выходной режим работы
                                     // для контакта светодиода
   pinMode(inputPin, INPUT_PULLUP); // Задаем входной режим работы
                                     // для контакта кнопки
}

void loop()
{
   int val = digitalRead(inputPin);     // Считываем входное значение
   if (val == LOW) // Нажатие кнопки устанавливает низкий уровень (LOW)
                   // на соответствующем контакте
   {
      digitalWrite(LED_BUILTIN, HIGH); // Если кнопка нажата, включаем светодиод
   }
}
```

Обсуждение работы решения и возможных проблем

Возвращаемое функцией `digitalRead()` значение проверяется с помощью оператора условия `if`. В скобках этого оператора указывается условие, которое может или выполняться или не выполняться. В первом случае оператор возвращает значение `true` (истина), а во втором — значение `false` (ложь). В приведенном примере условие указано как `val == LOW` (значение переменной `val` должно равняться значению `LOW`), и блок кода, следующий после оператора, исполняется только при выполнении этого условия. Под блоком кода здесь понимается весь код внутри фигурных скобок. В случае отсутствия фигурных скобок блоком кода будет просто следующий исполняемый оператор, завершающийся точкой с запятой.

Если требуется исполнить одно действие при выполнении условия и другое в противном случае, используется оператор `if...else`. Пример использования этого оператора приводится в листинге 2.20...

Листинг 2.20. Пример использования оператора условия `if...else`

```
/*
* Скетч Pushbutton
* Нажатие кнопки, подключенной к цифровому контакту 2 платы,
  включает встроенный светодиод
*/

const int inputPin = 2; // Контакт для подключения кнопки

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);      // Задаем выходной режим работы
                                      // для контакта светодиода
   pinMode(inputPin, INPUT_PULLUP);   // Задаем входной режим работы
                                      // для контакта кнопки
}

void loop()
{
   int val = digitalRead(inputPin);    // Считываем входное значение
   if (val == LOW)    // Нажатие кнопки устанавливает низкий уровень (LOW)
                      // на соответствующем контакте
   {
      // do this if val is LOW
      digitalWrite(LED_BUILTIN, HIGH); // Если кнопка нажата, включаем светодиод
   }
   else
   {
      // в противном случае, если значение val не равно LOW
      digitalWrite(LED_BUILTIN, LOW); // Если кнопка не нажата, выключаем светодиод
   }
}
```

Дополнительная информация

Булев тип данных рассматривается более подробно в *разд. 2.2*.

2.13. Циклическое исполнение последовательности операторов

ЗАДАЧА

Требуется циклически выполнять блок операторов, пока удовлетворяется условие.

РЕШЕНИЕ

Цикл `while` итерирует исполнение одной или нескольких команд, пока удовлетворяется заданное условие. В листинге 2.21 приводится пример скетча с использованием этого оператора.

Листинг 2.21. Использование оператора цикла `while`

```
/*
* Скетч Repeat
* Мигает светодиодом, пока удовлетворяется условие
*/

const int sensorPin = A0; // Ввод на аналоговый контакт A0

void setup()
{
   Serial.begin(9600);
   pinMode(LED_BUILTIN, OUTPUT); // Задаем режим вывода для контакта
                                 // встроенного светодиода
}

void loop()
{
   while(analogRead(sensorPin) > 100)
   {
      blink(); // Вызываем функцию мигания светодиодом
      Serial.print(".");
   }
   Serial.println(analogRead(sensorPin)); // Этот код не исполняется
            // до тех пор, пока цикл while не завершит исполнение!!!
}

void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
```

```
   delay(100);
   digitalWrite(LED_BUILTIN, LOW);
   delay(100);
}
```

В этом скетче блок кода внутри фигурных скобок будет исполняться в течение всего времени, пока выдаваемое функцией `analogRead()` значение больше чем 100. Светодиод выключен, пока величина выдаваемого датчиком значения равна или меньше 100, а когда она становится выше этого значения, светодиод начинает постоянно мигать. Такой скетч можно использовать, например, для тревожного мигания светодиодом, если какой-либо параметр процесса превышает пороговое значение.

Обсуждение работы решения и возможных проблем

Код, который циклически исполняется в результате удовлетворения условия цикла `while`, определяется фигурными скобками. При отсутствии фигурных скобок циклически исполняется только первая строка кода, следующая за оператором `while`, как демонстрируется в следующем фрагменте кода:

```
while(analogRead(sensorPin) > 100)
   blink(); // Исполняется строка кода, следующая сразу же
            // за выражением условия оператора while
   Serial.print("."); // Этот код исполняется только после завершения
                      // исполнения цикла while!!!
```

Цикл `do...while` подобен циклу `while`, но инструкции в блоке кода этого цикла исполняются перед тем, как проверяется условие. Эта форма цикла `while` используется в тех случаях, когда нужно, чтобы код исполнился хотя бы один раз, даже если условие не удовлетворяется. Формат цикла `do...while` приводится в листинге 2.22.

Листинг 2.22. Формат цикла `do...while`

```
do
{
   blink(); // Вызываем функцию мигания светодиодом
   Serial.print(".");
}
while (analogRead(sensorPin) > 100);
```

Этот код как минимум один раз мигнет светодиодом и будет мигать им до тех пор, пока датчик выдает значение больше чем 100. Но если значение не больше чем 100, то будет выполнено только одно мигание. Такой код можно применить, например, в схеме зарядки аккумуляторов, вызывая его каждые 10 секунд или около того. Одно мигание будет означать, что выполняется зарядка, а постоянное мигание, что аккумулятор полностью заряжен.

Код в цикле `while` будет исполняться до тех пор, пока не выполнится условие, вызывающее прекращение его исполнения, не позволяя исполнение никакого другого кода. Если исполнение цикла необходимо прервать в ответ на какое-либо другое условие, например тайм-аут, состояние датчика или тот или иной другой входной сигнал, это можно сделать с помощью оператора `break`, как демонстрируется в листинге 2.23.

Листинг 2.23. Использование оператора break

```
while(analogRead(sensorPin) > 100)
{
   blink();
   Serial.print(".");
   if(Serial.available())
   {
      while(Serial.available())
      {
         // Принимаем любые текущие входные данные
         Serial.read();
      }
      break; // Любые входные данные вызывают преждевременное завершение
               цикла
      }
}
```

Дополнительная информация

Смотрите также *главы 4* и *5*.

2.14. Повторение исполнения кода с использованием счетчика

ЗАДАЧА

Необходимо исполнить один или несколько операторов определенное количество раз.

РЕШЕНИЕ

Такая задача решается с помощью цикла `for`. Цикл `for` подобен циклу `while`, но его условия запуска и прекращения более управляемые.

Пример использования цикла `for` приводится в листинге 2.24. Этот скетч ведет счет от 0 до 3 и выводит в окно монитора порта значение переменной счетчика `i`.

Листинг 2.24. Пример использования цикла for

```
/*
Скетч ForLoop
Демонстрирует работу цикла for
*/

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

void loop()
{
   Serial.println("for(int i=0; i < 4; i++)");
   for(int i=0; i < 4; i++)
   {
      Serial.println(i);
   }
   delay(1000);
}
```

В результате исполнения этого скетча в окно монитора порта будут бесконечно выводиться следующие данные:

```
for(int i=0; i < 4; i++)
0
1
2
3
```

Обсуждение работы решения и возможных проблем

Определение цикла `for` состоит из трех частей: инициализации счетчика, проверки условия и итерации (итерация — это оператор, который исполняется в конце каждого прохода через цикл). Все части определения цикла разделены запятыми. В листинге 2.24 код `int i=0;` инициализирует переменную счетчика `i` начальным значением 0, код `i < 4;` проверяет, что значение переменной `i` меньше 4, а код `i++` инкрементирует переменную счетчика `i`.

В качестве счетчика можно использовать или уже существующую переменную, или же создать ее в коде оператора `for`. Версия скетча в листинге 2.25 использует переменную счетчика `j`, созданную в скетче ранее.

Листинг 2.25. Использование переменной счетчика, созданной в скетче ранее

```
int j;
Serial.println("for(j=0; j < 4; j++)");
```

```
for(j=0; j < 4; j++ )
{
   Serial.println(j);
}
```

Это практически такой же код, как и в листинге 2.24, с тем исключением, что в части инициализации переменной счетчика отсутствует ключевое слово `int`, поскольку тип этой переменной уже был определен ранее. Данные, выводимые в окно монитора порта этой версией, также практически такие же, что и данные первой версии:

```
for(j=0; j < 4; j++)
0
1
2
3
```

Управление прекращением исполнения цикла осуществляется в части проверки условия. В приведенном примере для этого выполняется проверка, что значение переменной счетчика меньше 4, и при невыполнении этого условия исполнение цикла завершается.

Если переменной счетчика присваивается начальное значение 0 и нужно, чтобы цикл исполнился, например, четыре раза, надо проверять, что значение переменной счетчика меньше 4. Цикл повторяется, пока соблюдается это условие, которому отвечают четыре значения при начальном значении, равном 0.

Код в листинге 2.26 проверяет, что значение переменной счетчика меньше или равно 4, и выводит в окно монитора порта цифры от 0 до 4.

Листинг 2.26. Проверка, что значение переменной счетчика меньше или равно 4

```
Serial.println("for(int i=0; i <= 4; i++)");
for(int i=0; i <= 4; i++)
{
   Serial.println(i);
}
```

Третьей частью определения цикла `for` является оператор итерации, который исполняется в конце каждого прохода через цикл. Это может быть любой действительный оператор языков C/C++. В примере в листинге 2.27 демонстрируется увеличение значения переменной счетчика `i` на 2 при каждой итерации цикла.

Листинг 2.27. Увеличение значения переменной счетчика цикла на 2

```
Serial.println("for(int i=0; i < 4; i+= 2)");
for(int i=0; i < 4; i+=2)
{
   Serial.println(i);
}
```

Эта версия скетча выводит в окно монитора порта только значения 0 и 2.

Выражение итерации может также вести счет в обратном направлении — т. е. от большего значения к меньшему. В листинге 2.28 показан пример убывающего счета от 3 до 0.

Листинг 2.28. Убывающий счет от 3 до 0

```
Serial.println("for(int i=3; i >= 0 ; i--)");
for(int i=3; i >= 0 ; i--)
{
   Serial.println(i);
}
```

Любую часть определения цикла `for` можно пропустить, включая и выражение итерации. Но при этом разделяющая точка с запятой должна присутствовать даже для пропущенной части определения.

В листинге 2.29 приводится пример цикла `for`, в котором в определении цикла отсутствует выражение итерации. Инкрементация переменной счетчика выполняется в теле цикла `for` при условии наличия входного сигнала высокого уровня на контакте 2.

Для работы этого цикла нужно предварительно задать входной режим работы для контакта 2 с помощью функции `pinMode()`.

Листинг 2.29. Цикл `for` без выражения итерации

```
pinMode(2, INPUT_PULLUP); // Эта часть содержится в функции setup()
/* ... */
Serial.println("for(int i=0; i < 4; )");
for(int i=0; i < 4; )
{
   Serial.println(i);
   if(digitalRead(2) == LOW)
   {
      i++; // Счетчик инкрементируется только в результате нажатия кнопки
   }
}
```

В этом примере значение переменной счетчика `i` модифицируется не в определении цикла `for`, а в теле кода цикла в операторе условия `if`.

Дополнительная информация

Дополнительная информация по циклу `for` приводится на странице справки вебсайта Arduino (**https://oreil.ly/dL5vO**).

2.15. Преждевременное завершение цикла

ЗАДАЧА

Требуется преждевременно выйти из цикла на основании какого-либо проверяемого условия.

РЕШЕНИЕ

Преждевременное завершение цикла осуществляется с помощью оператора `break`, пример использования которого приводится в листинге 2.30.

Листинг 2.30. Использование оператора `break`

```
/*
* Скетч break
* Демонстрирует использование оператора break
*/

const int switchPin = 2;  // Подключаем кнопку к цифровому вводу 2

void setup()
{
   Serial.begin(9600);
   pinMode(LED_BUILTIN, OUTPUT);     // Задаем режим вывода
                     // для контакта встроенного светодиода
   pinMode(switchPin, INPUT_PULLUP); // Задаем режим ввода для контакта кнопки
}

void loop()
{
   while(true) // Бесконечный цикл
   {
      if(digitalRead(switchPin) == LOW)
      {
         break; // Прерываем исполнение цикла при нажатии кнопки
      }
      blink(); // Вызываем функцию мигания светодиодом
   }
}

void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
   delay(100);
   digitalWrite(LED_BUILTIN, LOW);
   delay(100);
}
```

Обсуждение работы решения и возможных проблем

Код этого примера похож на код циклов `while`, но в нем используется оператор `break` для преждевременного завершения цикла при наличии входного сигнала низкого уровня на указанном цифровом контакте. Такой низкий уровень может подаваться на этот контакт при нажатии кнопки, как показано в схеме на рис. 5.5 в *разд. 5.2*. В результате исполнение цикла будет завершено и светодиод перестанет мигать, несмотря на то, что условие для его исполнения продолжает удовлетворяться.

Дополнительная информация

Дополнительная информация по оператору `break` приводится на странице справки веб-сайта Arduino (**https://oreil.ly/-6MsQ**).

Прерывание представляет собой встроенную аппаратную возможность микроконтроллера, позволяющую предпринимать более или менее немедленное действие в ответ на изменение входного состояния контакта. Более подробная информация на эту тему приводится в *разд. 18.2*.

2.16. Реагирование на множественные состояния одной переменной

ЗАДАЧА

Требуется выполнить то или иное действие в зависимости от одного из многих состояний одной переменной. Хотя можно было бы использовать несколько операторов `if` и `if...else`, код такого решения будет слишком сложным и трудным для понимания и модифицирования. Кроме этого, возможно, потребуется проверить некоторый диапазон значений.

РЕШЕНИЕ

Эта задача решается с помощью оператора `switch`, который позволяет делать выбор действия на основании одного из нескольких возможных значений. Его функциональность похожа на функциональность нескольких операторов `if/if...else`, но он имеет более краткий формат. Пример использования этого оператора приводится в листинге 2.31.

Листинг 2.31. Пример использования оператора `switch`

```
/*
* Скетч SwitchCase
* Скетч примера использования оператора switch
  Выполняет разные действия в зависимости от полученного
  по последовательному интерфейсу символа
*
```

```
* При получении символа "1" мигает светодиодом один раз, символа "2" – два раза
* Символ "+" включает светодиод, символ "-" – выключает
* Любой другой символ выводит сообщение в окно монитора порта
*/

void setup()
{
   Serial.begin(9600); // Задаем скорость обмена по последовательному порту 9600 бод
   pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
   if ( Serial.available()) // Проверяем наличие хотя бы одного принятого символа
   {
      char ch = Serial.read();
      switch(ch)
      {
         case '1':
            blink();
            break;
         case '2':
            blink();
            blink();
            break;
         case '+':
            digitalWrite(LED_BUILTIN, HIGH);
            break;
         case '-':
            digitalWrite(LED_BUILTIN, LOW);
            break;
         case '\n': // Новая строка, можно игнорировать
            break;
         case '\r': // Возврат каретки, можно игнорировать
            break;
         default:
            Serial.print(ch);
            Serial.println(" was received but not expected");
                        // получен, но не ожидался.
            break;
      }
   }
}

void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
   delay(500);
```

```
    digitalWrite(LED_BUILTIN, LOW);
    delay(500);
}
```

Обсуждение работы решения и возможных проблем

Оператор `switch` проверяет значение переменной `ch`, полученное по последовательному интерфейсу, и переходит к метке, совпадающей с этим значением. Метки должны быть числовыми константами. Это возможно, поскольку значения типа `char` представляются числовыми кодами (см. *разд. 2.2*). Но использовать строки в качестве меток нельзя. Все метки также должны быть разными. Если после последнего рабочего оператора метки не следует оператор `break`, исполнение будет продолжено в следующей метке, как демонстрируется в фрагменте кода, приведенном в листинге 2.32.

Листинг 2.32. Пример отсутствия оператора `break` в коде метки

```
case '1':
    blink(); // В конце кода метки отсутствует оператор break
case '2':
    blink(); // Исполнении кода метки 1 переходит сюда
    blink();
    break; // Оператор break вызывает выход из оператора switch
```

Если в конце кода метки отсутствует оператор `break` (как показано в примере, приведенном в листинге 2.32), исполнение кода продолжается в следующую метку. В этом примере, несмотря на то, что скетчу был передан символ '1', будет выполнено три мигания светодиодом. Случайный пропуск оператора `break` в коде метки — это распространенная ошибка. Но иногда бывает полезным не вставлять этот оператор преднамеренно. Тем не менее такой подход может сбить с толку других пользователей, работающих с кодом, поэтому следует комментировать причины такого пропуска.

Если оператор `switch` не работает должным образом, проверьте, что вы не забыли вставить оператор `break` в код какой-либо его метки.

Метка `default` предназначена для обработки всех остальных случаев, которые не попадают под остальные метки. Если метка `default` не используется, при отсутствии совпадения значения условия с какой-либо меткой не выполняется никакое действие.

Дополнительная информация

Дополнительная информация по оператору `switch` приводится на странице справки веб-сайта Arduino (**https://oreil.ly/pPLn-**).

2.17. Сравнение символьных и числовых значений

ЗАДАЧА

Требуется сравнить два значения.

РЕШЕНИЕ

Для сравнения целочисленных значений используются операторы сравнения, приведенные в табл. 2.5.

***Таблица 2.5.** Операторы сравнения*

Оператор	Выполняемое сравнение	Пример
==	Равно	`2 == 3` // результат будет `false` (ложь)
!=	Не равно	`2 != 3` // результат будет `true` (истина)
>	Больше чем	`2 > 3` // результат будет `false` (ложь)
<	Меньше чем	`2 < 3` // результат будет `true` (истина)
>=	Больше чем или равно	`2 >= 3` // результат будет `false` (ложь)
<=	Меньше чем или равно	`2 <= 3` // результат будет `true` (истина)

Скетч в листинге 2.33 демонстрирует использование операторов сравнения.

Листинг 2.33. Использование операторов сравнения

```
/*
* Скетч RelationalExpressions
* Демонстрирует работу операторов сравнения
*/

int i = 1; // Значения для сравнивания
int j = 2;

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

void loop()
{
   Serial.print("i = ");
   Serial.print(i);
   Serial.print(" and j = ");
   Serial.println(j);
```

```
   if(i < j)
   Serial.println(" i is less than j");
                // i меньше чем j
   if(i <= j)
   Serial.println(" i is less than or equal to j");
                // i меньше чем или равно j
   if(i != j)
   Serial.println(" i is not equal to j");
                // i не равно j
   if(i == j)
   Serial.println(" i is equal to j");
                // i равно j
   if(i >= j)
   Serial.println(" i is greater than or equal to j");
                // i больше чем или равно j
   if(i > j)
   Serial.println(" i is greater than j");
                // i больше чем j
   Serial.println();
   i = i + 1;
   if(i > j + 1)
   {
      delay(10000); // Длинная пауза когда разница между i и j больше 1
   }
   else
   {
      delay(1000); // Короткая пауза
   }
}
```

Результат исполнения этого скетча приводится в листинге 2.34.

Листинг 2.34. Результат исполнения скетча из листинга 2.33

```
i = 1 and j = 2
   i is less than j
   i is less than or equal to j
   i is not equal to j

i = 2 and j = 2
   i is less than or equal to j
   i is equal to j
   i is greater than or equal to j

i = 3 and j = 2
   i is not equal to j
   i is greater than or equal to j
   i is greater than j
```

Обсуждение работы решения и возможных проблем

Обратите внимание на то, что в качестве символа оператора проверки на равенство используются два знака равенства: ==. Одной из наиболее распространенных ошибок программирования является ошибочное использование вместо оператора сравнения (==) оператора присваивания, который обозначается одним символом знака равенства (=).

В следующей строке кода программист хочет предпринять какое-либо действие, если значение переменной `i` равно 3:

```
if(i == 3) // Проверяем, равно ли i числу 3
```

Но предположим, что для этой же задачи он использовал следующий код:

```
if(i = 3) // Ошибочно использован один знак равенства !!!!
```

Этот код всегда будет возвращать значение `true` (истина), поскольку переменной `i` будет присвоено значение 3, и при сравнении оба эти значения будут одинаковыми.

Операторы сравнения можно также использовать для сравнения символов, поскольку они представляются числовыми кодами ASCII (см. *разд. 2.2*). Результатом следующего сравнения будет `true` (истина):

```
if ('b' > 'a')
```

Это сравнение также даст результат `true`, поскольку символ `'a'` (английская буква "a") имеет числовое значение 97 кода ASCII:

```
if ('a' == 97)
```

Но строки нельзя напрямую сравнивать с числовыми значениями, как демонстрируется в фрагменте кода, приведенном в листинге 2.35.

Листинг 2.35. Попытки сравнения строки с числовыми значениями

```
String word1 = String("Hello");
char word2[] = "World";
if (word1 > 'G') // Этот код вызовет ошибки при компиляции
{
   Serial.println("word1 > G");
}
if (word2 >= 'W') // Этот код также вызовет ошибки при компиляции
{
   Serial.println("word2 >= W");
}
```

Однако всегда можно сравнить число или отдельный символ с одним символом строки, как демонстрируется в фрагменте кода, приведенном в листинге 2.36.

Листинг 2.36. Сравнение отдельного символа с одним символом строки

```
if (word1.charAt(0) > 'G')
{
   Serial.println("word1[0] > G");
}
if (word2[0] >= 'W')
{
   Serial.println("word2[0] >= W");
}
```

Дополнительная информация

Более подробная информация по операторам сравнения предлагается на странице справки веб-сайта Arduino (**https://oreil.ly/CZ4U0**).

Таблицу кодов символов ASCII можно найти здесь: **https://oreil.ly/uNpYu**.

2.18. Сравнение строк

ЗАДАЧА

Требуется выполнить проверку двух строк на идентичность.

РЕШЕНИЕ

Для сравнения двух строчных значений используется функция `strcmp()`. В листинге 2.37 приводится фрагмент кода, демонстрирующий ее использование.

Листинг 2.37. Использование функции `strcmp()`

```
char string1[ ] = "left";
char string2[ ] = "right";
if(strcmp(string1, string2) == 0)
{
   Serial.println("strings are equal"); // строки одинаковые
}
```

Обсуждение работы решения и возможных проблем

Если строки одинаковые, функция `strcmp()` возвращает значение 0. Если при первом несовпадении символ первой строки имеет большее значение, чем символ второй строки, функция возвращает значение больше чем 0. Если значение первого несовпадающего символа первой строки меньше, чем символа второй строки, функция возвращает значение меньше чем 0. Обычно требуется только узнать, являются ли строки одинаковыми, и хотя проверка на ноль может с первого взгляда казаться

неинтуитивной, вскоре вырабатывается привычка воспринимать этот результат должным образом.

Имейте в виду, что строки неодинаковой длины не будут равными, даже если короткая строка входит в длинную. Соответственно,

```
if (strcmp("left", "leftcenter") == 0) // Результатом этого сравнения
                                       // будет false (ложь)
```

С помощью функции `strncmp()` можно сравнивать только определенное число символов в обеих строках. Количество символов для сравнения передается в третьем параметре, и сравнение остальных символов не выполняется:

```
if (strncmp("left", "leftcenter", 4) == 0) // Результатом этого сравнения
                                           // будет true (истина)
```

В отличие от строк массивов символов, строки объекта `String` можно сравнивать напрямую, как демонстрируется в фрагменте кода, приведенном в листинге 2.38.

Листинг 2.38. Сравнение строк объекта `String`

```
String stringOne = String("this");
if (stringOne == "this")
{
   Serial.println("this will be true"); // Результат будет true
}
if (stringOne == "that")
{
   Serial.println("this will be false"); // Результат будет false
}
```

Дополнительная информация по сравнению строк приводится на веб-странице учебного пособия Arduino (**https://oreil.ly/M48_D**).

Дополнительная информация

Дополнительная информация по функции `strcmp()` приводится на веб-сайте cplusplus (**https://oreil.ly/rkoUk**). Использование объекта строки `String` было представлено в *разд. 2.5*.

2.19. Логические сравнения

ЗАДАЧА

Требуется определить логическое соответствие между двумя или более выражениями. Например, мы хотим выполнить то или иное действие на основании нескольких значений оператора `if`.

РЕШЕНИЕ

Логические сравнения осуществляются с помощью логических операторов, список которых приводится в табл. 2.6.

***Таблица 2.6.** Логические операторы*

Обозначение	Функция	Описание
&&	Логическое И	Возвращает true (истина), если выражения по обеим сторонам оператора && есть true
||	Логическое ИЛИ	Возвращает true (истина), если выражение хотя бы на одной стороне оператора || есть true.
!	НЕ	Возвращает true, если выражение false, или false, если выражение true

Обсуждение работы решения и возможных проблем

Логические операторы возвращают булево значение `true` или `false` в зависимости от результатов оценки логической взаимосвязи. Для рассматриваемых здесь примеров предполагается подключение двух датчиков (кнопок) к цифровым контактам 2 и 3 согласно схеме, приведенной на рис. 5.5 в *разд. 5.2* (при этом контакты настроены для работы в режиме ввода подключением внутренних повышающих резисторов).

В следующем коде логическая операция И (`&&`) возвратит `true` только тогда, когда значение обоих операндов будет `true`, и `false` в противном случае:

```
if(digitalRead(2) && digitalRead(3))
   blink(); // Мигаем светодиодом, если на обоих контактах присутствует
            // высокий логический уровень
```

А в этом коде логическая операция ИЛИ (`||`) возвратит `true`, если значение любого из операндов будет `true`, и `false`, если значение обоих операндов `false`:

```
if( digitalRead(2) || digitalRead(3) )
  blink(); // Мигаем светодиодом, если входной высокий уровень
           // присутствует на любом из контактов (или на обоих)
```

Логический оператор НЕ (`!`) оперирует только с одним операндом, инвертируя его значение. Если значение операнда `true`, оператор возвращает `false`, а если `false` — возвращает `true`.

```
if( !digitalRead(2) )
   blink(); // Мигаем светодиодом, если на контакте отсутствует
            // входной высокий уровень
```

2.20. Операции с битами

ЗАДАЧА

Требуется устанавливать или обнулять определенные биты двоичного значения.

РЕШЕНИЕ

Эта задача решается с помощью побитовых операторов, список которых приводится в табл. 2.7. Префикс `0b` означает двоичное представление числа, чтобы различать в таблице десятичные и двоичные числа.

Таблица 2.7. *Побитовые операторы*

Символ	Функция	Результат действия	Пример
`&`	Побитовая операция И	Устанавливает биты (присваивает им значение 1) результата, если оба соответствующих исходных бита установлены. В противном случае обнуляет биты результата	`3 & 1` равняется 1 (0b11 & 0b01 равняется 0b01)
`|`	Побитовая операция ИЛИ	Устанавливает биты результата, если установлен хотя бы один из соответствующих исходных битов	`3 | 1` равняется 3 (0b11 | 0b01 равняется 0b11)
`^`	Побитовая операция Исключающее ИЛИ (ИСКЛ-ИЛИ)	Устанавливает биты результата, если установлен только один из соответствующих исходных битов	`3 ^ 1` равняется 2 (0b11 ^ 0b01 равняется 0b10)
`~`	Побитовая операция отрицания (НЕ)	Инвертирует значение каждого бита. Результат зависит от количества битов в типе данных	`~1` равняется 254 (-00000001 равняется 11111110)

В листинге 2.39 приводится пример скетча, демонстрирующего использование побитовых операторов.

Листинг 2.39. Демонстрация использования побитовых операторов

```
/*
* Скетч bits
* Демонстрирует использование побитовых операторов
*/

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

void loop()
{
   Serial.print("3 & 1 equals "); // Побитовое И над 3 и 1
   Serial.print(3 & 1); // Выводим результат в монитор порта
   Serial.print(" decimal, or in binary: ");
   Serial.println(3 & 1 , BIN); // выводим двоичное представление результата
```

```
    Serial.print("3 | 1 equals "); // Побитовое ИЛИ над 3 и 1
    Serial.print(3 | 1 );
    Serial.print(" decimal, or in binary: ");
    Serial.println(3 | 1 , BIN); // Выводим двоичное представление результата

    Serial.print("3 ^ 1 equals "); // Побитовое ИСКЛ-ИЛИ над 3 и 1
    Serial.print(3 ^ 1); Serial.print(" decimal, or in binary: ");
    Serial.println(3 ^ 1 , BIN); // Выводим двоичное представление результата

    byte byteVal = 1;
    int intVal = 1;

    byteVal = ~byteVal; // Побитовая операция отрицания (НЕ)
    intVal = ~intVal;

    Serial.print("~byteVal (1) equals "); // Побитовое НЕ
                                          // на 8-битном значении
    Serial.println(byteVal, BIN); // Выводим двоичное представление результата
    Serial.print("~intVal (1) equals "); // Побитовое НЕ
                                         // на 16-битном значении
    Serial.println(intVal, BIN); // Выводим двоичное представление результата

    delay(10000);
}
```

В окне монитора порта должен выводиться следующий результат:

```
3 & 1 equals 1 decimal, or in binary: 1
3 | 1 equals 3 decimal, or in binary: 11
3 ^ 1 equals 2 decimal, or in binary: 10
~byteVal (1) equals 11111110
~intVal (1) equals 11111111111111111111111111111110
```

Обсуждение работы решения и возможных проблем

Побитовые операторы используются для установки или проверки значения битов. При выполнении операции и или или над двумя значениями оператор обрабатывает индивидуально каждую отдельную пару битов. Понять работу этих операторов будет легче, если рассматривать, как они работают с двоичными представлениями значений.

Например, десятичное целое число 3 — это двоичное 11, а десятичное целое 1 — это двоичное 1. Побитовая операция и обрабатывает индивидуально каждую отдельную пару битов этих исходных значений. Самый правый бит обоих значений равен 1, поэтому результат операции и над этой парой битов будет 1. Следующая левая пара битов будет 1 и 0. Результат операции и над этой парой битов будет 0. Оба бита последней пары битов равны 0, поэтому результат операции и над этой парой битов будет 0. Иными словами, если значение обоих битов каждой пары равно 1, результат будет 1, в противном же случае результат будет 0. Таким образом,

`11 & 01` равняется 01. Двоичные числа часто записываются с ведущими нулями, поскольку это помогает сразу же понять результат побитовой операции, но ведущие нули не являются значащими.

В табл. 2.8–2.10 приводятся результаты операций И, ИЛИ и ИСКЛ-ИЛИ над одной парой битов, что должно помочь понять работу этих операторов.

Таблица 2.8. *Побитовая операция И*

Бит 1	Бит 2	Бит 1 И Бит 2
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 2.9. *Побитовая операция ИЛИ*

Бит 1	Бит 2	Бит 1 ИЛИ Бит 2
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 2.10. *Побитовая операция ИСКЛ-ИЛИ*

Бит 1	Бит 2	Бит 1 ^ Бит 2
0	0	0
0	1	1
1	0	1
1	1	0

Все побитовые операторы берут два операнда, за исключением оператора отрицания. Оператор отрицания просто меняет значения каждого бита своего операнда на обратное, т. е. 0 становится 1, а 1 — 0. Например, результатом операции отрицания над значением типа `byte` (8-битовым значением) 0000 0001 будет 1111 1110. Значения типа `int` состоят из 2 байтов, или 16 битов, поэтому результат операции отрицания над значением 0000 0000 0000 0001 будет 1111 1111 1111 1110.

Дополнительная информация

Дополнительная информация по побитовым операторам приводится на странице справки веб-сайта Arduino (**https://oreil.ly/qSJEx**).

2.21. Комбинирование операции и присваивания

ЗАДАЧА

Понять, как использовать составные операторы. В коде часто используются выражения, выполняющие несколько операций в одном операторе. Например, `a += b`, `a >>= b` или `a &= b`. Мы хотим разобраться, как именно работают такие составные операторы.

РЕШЕНИЕ

В табл. 2.11 приводится список составных операторов и их эквивалентные полные формы.

Таблица 2.11. *Составные операторы*

Оператор	Пример	Эквивалентная полная форма
`+=`	`значение += 5;`	`значение = значение + 5; // к значению добавляется 5`
`-=`	`значение -= 4;`	`значение = значение -4; // из значения вычитается 4`
`*=`	`значение *= 3;`	`значение = значение * 3; // значение умножается на 3`
`/=`	`значение /= 2;`	`значение = значение / 2; // значение делится на 2`
`>>=`	`значение >>= 2;`	`значение = значение >> 2; // биты значения сдвигаются вправо на две позиции`
`<<=`	`значение <<= 2;`	`значение = значение << 2; // биты значения сдвигаются влево на две позиции`
`&=`	`маска &= 2;`	`маска = маска & 2; // операция И над битами маски и значения 2`
`|=`	`маска |= 2;`	`маска = маска | 2; // операция ИЛИ над битами маски и значения 2`

Обсуждение работы решения и возможных проблем

Во время исполнения программы составные операторы не предоставляют никакого преимущества в производительности над эквивалентными полными выражениями, но могут сбивать с толку начинающих программистов. Однако опытные программисты часто используют составные операторы в исходном коде, поэтому полезно научиться распознавать такие операторы, когда придется с ними столкнуться.

Дополнительная информация

Дополнительная информация по составным операторам предлагается на веб-сайте Arduino на соответствующих веб-страницах справки. На домашней странице справки (**https://oreil.ly/z-1_h**) имеется указатель на страницы справки по отдельным составным операторам.

ГЛАВА 3

Математические операции

3.0. Введение

Практически в каждом скетче используются математические операции для манипулирования значениями переменных. В этой главе приводится краткий обзор наиболее распространенных математических операций. Те, кто уже знаком с языком программирования C или C++, могут испытывать соблазн пропустить эту главу, но мы рекомендуем просмотреть ее, поскольку вам могут встретиться некоторые выражения, которые вы никогда не использовали, но популярные у программистов на Arduino (например, применение функции `bitSet()` для изменения значения бита). Читатели же, которым язык C/C++ совершенно незнаком, могут получить начальные сведения о нем в одной из книг, упомянутых в предисловии.

3.1. Сложение, вычитание, умножение и деление

ЗАДАЧА

Вам нужно выполнять в скетче простые математические операции со значениями. Вы хотите управлять порядком, в котором выполняются операции, и вам может потребоваться обработка различных типов переменных.

РЕШЕНИЕ

Пример решения этой задачи для целочисленных значений приводится в листинге 3.1.

Листинг 3.1. Основные математические операции с целочисленными значениями

```
int myValue;
myValue = 1 + 2;    // Сложение
myValue = 3 - 2;    // Вычитание
myValue = 3 * 2;    // Умножение
myValue = 3 / 2;    // Деление (результат равен 1)
```

Обсуждение работы решения и возможных проблем

Операции сложения, вычитания и умножения с целыми числами в целом работают ожидаемым образом.

При делении же одного целого числа на другое целое число также получается целое число, а дробный остаток отбрасывается, как показано в приведенном примере, — значение переменной `myValue` после деления будет равно 1 (операция деления, когда требуется дробный результат, рассматривается в *разд. 2.3*).

Результаты выполнения сложных выражений (наподобие следующего), которые включают несколько разных операций, могут показаться неоднозначными:

```
int value = 1 + 2 * 3 + 4;
```

но каждая операция выполняется в строго установленном порядке. В частности, умножение и деление выполняются перед сложением и вычитанием, поэтому в результате получится 11. Чтобы избежать возможных неясностей и сделать более понятным порядок операций, в коде рекомендуется использовать скобки. Например, результат выражения:

```
int value = 1 + (2 * 3) + 4;
```

будет таким же, что и предыдущего, но в нем ясно виден порядок операций.

С помощью скобок можно также указать и требуемый порядок операций, как показано в следующем примере:

```
int value = ((1 + 2) * 3) + 4;
```

В результате выполнения этих операций получится 13. Вычисления во внутренних скобках выполняются первыми, поэтому сначала к 1 добавляется 2, результат этого сложения (3) умножается на 3, а к полученному произведению (9) добавляется 4, что дает конечный результат 13.

Нужно принимать меры, чтобы полученный результат мог поместиться в свою переменную, поскольку компилятор Arduino по умолчанию не предупреждает о возможных проблемах подобного рода (предупреждения можно включить, выполнив команду меню **Файл | Настройки** и выбрав в выпадающем списке **Предупреждения компилятора** опцию **Все**. Более подробно этот вопрос рассматривается в *разд. 2.2*). Но, даже указав правильный тип переменной, все равно можно получить переполнение результата, как демонстрируется в следующем коде:

```
// 60 секунд в минуте, 60 минут в часе, 24 часа в дне
long seconds_per_day = 60 * 60 * 24;
```

Теоретически результат 86 400 должен поместиться в переменную `seconds_per_day`, т. к. для нее был указан тип `long`. Но в действительности в ней сохранится значение 20 864 — из-за двукратного переполнения этой переменной: 86,400 – 32,768 * 2 = = 20,864. Переполнение же происходит вследствие того, что компилятор C среды Arduino IDE присваивает этой переменной тип `int` — несмотря на объявленный тип `long`, — поскольку все операнды выражения имеют тип `int`. Чтобы избежать этого, компилятору нужно явно указать, что результат операций над переменными типа `int` должен быть типа `long`, добавив в конце значения выражения, которое будет обрабатываться первым, суффикс `L`:

```
long seconds_per_day = 60L * 60 * 24;
```

Не забывайте также, что при использовании скобок выражения внутри скобок обрабатываются первыми, поэтому следующий код снова вызовет переполнение:

```
long seconds_per_day_plus_one = 1L + 60 * (60 * 24);
```

А вот этот даст правильный результат:

```
long seconds_per_day_plus_one = 1 + 60 * (60L * 24);
```

Операции с числами с плавающей запятой подвержены всем погрешностям, описанным в *разд. 2.3*. Например, следующий код, который выводит результат деления 36,3 на 3 с точностью до 10 десятичных разрядов после запятой, будет отображать не ожидаемое значение 12,1, а 12,0999994277:

```
Serial.println(36.3/3, 10);
```

Точный результат можно отобразить с помощью приема, который рассматривается в *разд. 3.3*.

Дополнительная информация

Дополнительный материал по этим вопросам излагается в *разд. 2.2* и *2.3*.

3.2. Увеличение и уменьшение значений переменных

ЗАДАЧА

Требуется увеличить или уменьшить значение переменной.

РЕШЕНИЕ

Пример решения этой задачи приводится в листинге 3.2.

Листинг 3.2. Увеличение и уменьшение значений

```
int myValue = 0;

myValue = myValue + 1; // Увеличивает значение переменной myValue на 1
myValue += 1;          // Делает то же самое, что и предыдущая строка кода

myValue = myValue - 1; // Уменьшает значение переменной myValue на 1
myValue myValue -= 1;  // Делает то же самое, что и предыдущая строка кода

myValue = myValue + 5; // Увеличивает значение переменной myValue на 5
myValue += 5;          // Делает то же самое, что и предыдущая строка кода
```

Обсуждение работы решения и возможных проблем

Одной из самых распространенных задач при создании скетчей Arduino является увеличение и уменьшение значений переменных. Операторы языка Arduino делают легким выполнение этой задачи. Увеличение значения на 1 называется *инкремен-*

тированием, а уменьшение на 1 — *декрементированием*. Версия этой операции в развернутом формате выглядит следующим образом:

```
myValue = myValue + 1; // Увеличиваем значение переменной myValue на 1
```

Но оператор инкремента (или декремента) можно объединить с оператором присваивания, что даст версию сокращенного формата:

```
myValue += 1;     // Делает то же самое, что и предыдущая строка кода
```

В приведенных кратких форматах операции увеличения (или уменьшения) вместо 1 можно указать какое-либо другое значение. Но если выполняется операция инкремента или декремента (т. е. увеличения или уменьшения значения на 1), то формат операции можно сократить еще больше, используя операторы ++ и -- соответственно:

```
myValue++;        // Делает то же самое, что и предыдущая строка кода
```

Когда операторы сокращенного формата инкремента или декремента идут после переменной, их операция выполняется после возвращения значения переменной. Такие операторы называются *постинкрементными* и *постдекрементными*. Если же эти операторы предшествуют переменной, то их операция выполняется до возвращения значения переменной. Такие операторы называются *преинкрементными* и *предекрементными*. В листинге 3.3 приводится код, демонстрирующий работу постинкрементных и преинкрементных операторов.

Листинг 3.3. Демонстрация работы постинкрементных и преинкрементных операторов

```
int myVal = 1;
Serial.println(myVal++);        // Выводится значение 1
Serial.println(myVal);          // Выводится значение 2
Serial.println(++myVal);        // Выводится значение 3
Serial.println(myVal);          // Выводится значение 3
```

Дополнительная информация

Базовая информация по математическим операциям приводится в *разд. 3.1*.

3.3. Нахождение остатка от деления двух значений

ЗАДАЧА

Требуется определить остаток после деления двух целочисленных значений.

РЕШЕНИЕ

Эта задача решается с помощью оператора деления по модулю (символ %):

```
int myValue0 = 20 % 10; // Возвращает остаток после деления 20 на 10
int myValue! = 21 % 10; // Возвращает остаток после деления 21 на 10
```

В результате выполнения первой строки кода значение переменной `myValue0` будет 0 (при делении 20 на 10 остаток равен 0). А в результате выполнения второй строки кода значение переменной `myValue1` будет 1 (при делении 21 на 10 остаток равен 1).

Обсуждение работы решения и возможных проблем

Оператор *деления по модулю* оказывается на удивление полезным, особенно когда необходимо узнать, является ли одно число кратным какому-либо другому числу. Например, код приведенного в этом разделе решения можно улучшить, чтобы определять значения, кратные числу 10 (листинг 3.4).

Листинг 3.4. Определение кратности значения числу 10

```
for (int myValue = 0; myValue <= 100; myValue += 5)
{
   if (myValue % 10 == 0)
   {
      Serial.println("The value is a multiple of 10");
   }
}
```

Этот код выполняет деление по модулю на 10 значения переменной `myValue` и сравнивает полученный результат со значением 0 (см. *разд. 2.17*). Если результат равен 0, то выводится сообщение, что значение кратно 10.

В листинге 3.5 приводится похожий пример, в котором, выполняя деление числа по модулю на 2, можно по результату определить, четное ли это число или нечетное.

Листинг 3.5. Определение четности или нечетности значения

```
for (int myValue = 0; myValue <= 10; myValue++)
{
   if (myValue % 2 == 0)
   {
      Serial.println("Значение четное");
   }
   else
   {
      Serial.println("Значение нечетное");
   }
}
```

А следующий код (листинг 3.6) вычисляет, какой будет час по истечении заданного количества часов от текущего часа.

Листинг 3.6. Вычисление часа после определенного периода времени

```
void printOffsetHour(int hourNow, int offsetHours)
{
   Serial.println((hourNow + offsetHours) % 24);
}
```

Оператор деления по модулю также можно использовать для эмуляции операций с числами с плавающей запятой (точкой). Рассмотрим, например, задачу из *разд. 3.1*, где деление числа 36,3 на 3 дает результат 12,0999994277 вместо ожидаемого 12,1.

Чтобы исправить ситуацию, делимое и делитель сначала умножаются на 10, а затем выполняется операция деления, чтобы получить целую часть результата:

```
int int_part = 363/30; // результат: 12
```

Затем вычисляется остаток от деления этих двух чисел, результат умножается на 100, а полученное произведение делится на 3, чтобы получить дробную часть:

```
int remainder = 363 % 30; // результат: 3
int fractional_part = remainder * 100 / 30;
```

Наконец, отображаем в одну строку целую и дробную части, разделенные точкой, что дает нам 12.10:

```
Serial.print(int_part); Serial.print("."); Serial.println(fractional_part);
```

Дополнительная информация

Более подробная информация по оператору деления по модулю предлагается на странице справки веб-сайта Arduino (**https://oreil.ly/ffecV**).

3.4. Определение абсолютного значения числа

ЗАДАЧА

Требуется определить абсолютное значение числа.

РЕШЕНИЕ

Функция `abs(x)` возвращает абсолютное значение числа `x`. Код следующего примера (листинг 3.7) вычисляет абсолютное значение разницы между двумя показателями входных аналоговых сигналов (более подробная информация по функции `analogRead()` приводится в *главе 5*).

Листинг 3.7. Определение абсолютного значения числа

```
int x = analogRead(A0);
int y = analogRead(A1);
if (abs(x-y) > 10)
```

```
{
   Serial.println("Разница между аналоговыми значениями больше 10");
}
```

Обсуждение работы решения и возможных проблем

Оператор `abs(x-y);` возвращает абсолютное значение разницы между значениями переменных `x` и `y`. Функция `abs()` принимает в качестве параметров целочисленные значения (типа `int` или `long`). Информация о том, как получить абсолютное значение числа с плавающей точкой, приводится в *разд. 2.3*.

Дополнительная информация

Более подробная информация по оператору деления по модулю предлагается на странице справки веб-сайта Arduino (**https://oreil.ly/uDEs7**).

3.5. Ограничение диапазона значений числа

ЗАДАЧА

Требуется обеспечить, чтобы значение переменной было в пределах определенного диапазона значений.

РЕШЕНИЕ

Оператор `constrain(x, min, max)` сводит значение параметра `x` к диапазону значений в пределах, указанных в передаваемых ему параметрах `min` и `max`:

```
int myConstrainedValue = constrain(myValue, 100, 200);
```

Обсуждение работы решения и возможных проблем

Переменной `myConstrainedValue` всегда будет присваиваться значение, большее или равное 100 и меньшее или равное 200. Если значение переменной `myValue` меньше чем 100, результат будет 100, а если больше чем 200, результат будет 200.

В табл. 3.1 приводится несколько примеров выходных значений функции `constrain()` для пределов `min` и `max` величиной 100 и 200 соответственно.

***Таблица 3.1.** Значение, возвращаемое функцией constrain для пределов min = 100 и max = 200*

myValue (входное значение)	constrain(myValue, 100, 200)
99	100
100	100
150	150
200	200
201	200

Дополнительная информация

См. также *разд. 3.6.*

3.6. Определение меньшего или большего числа из двух или более чисел

ЗАДАЧА

Требуется определить меньшее или большее из двух значений.

РЕШЕНИЕ

Функция `min(x,y)` возвращает меньшее из двух чисел, а функция `max(x,y)` — большее.

В листинге 3.8 приводится пример использования этих функций.

Листинг 3.8. Определение меньшего и большего из двух чисел

```
int myValue = analogRead(A0);
int myMinValue = min(myValue, 200); // Значение myMinValue будет меньшим
                                    // из значений myValue и 200
int myMaxValue = max(myValue, 100); // Значение myMaxValue будет большим
                                    // из значений myValue и 100
```

Обсуждение работы решения и возможных проблем

В табл. 3.2 приводится несколько примеров выходных значений, возвращаемых функцией `min()`. Как можно видеть, возвращаемое значение всегда равно входному значению `myValue`, пока входное значение не превысит значение 200.

***Таблица 3.2.** Результаты функции `min(myValue, 200)`*

myValue (the input value)	min(myValue, 200)
99	99
100	100
150	150
200	200
201	200

В табл. 3.3 приводится несколько примеров выходных значений, возвращаемых функцией `max()`. Как можно видеть, возвращаемое значение всегда равно входному значению `myValue`, пока входное значение не станет меньше чем 100.

Таблица 3.3. Результаты функции `max(myValue, 100)`

myValue (the Input value)	max(myValue, 100)
99	100
100	100
150	150
200	200
201	201

Функцию `min()` можно использовать для задания верхнего предела диапазона значений, указываемого числовым параметром. На первый взгляд, использование функции `min()` для задания максимального предела может казаться нелогичным, но поскольку возвращаемое ею значение будет меньшим из значений параметра переменной и числового параметра, то оно никогда не превысит значение числового параметра (равное 200 в приведенном примере).

Подобным образом функцию `max()` можно использовать для задания нижнего предела диапазона значений. Возвращаемое ею значение никогда не будет меньшим, чем значение ее числового параметра (равное 100 в нашем примере).

Найти большее или меньшее число из более чем двух значений, можно, используя вложенные функции, когда результат функции служит параметром этой же функции, как показано в следующем примере:

```
// Переменной myMinValue будет присвоено меньшее из трех значений
// аналоговых показаний:
int myMinValue = min(analogRead(0), min(analogRead(1), analogRead(2)));
```

Здесь сначала определяется меньшее значение из значений первых двух аналоговых показаний, а затем меньшее из этого результата и значения третьего аналогового показания. Этот подход можно расширить до любого требуемого числа значений, но при этом следует быть внимательным, чтобы правильно расставлять скобки. В следующем примере определяется большее из четырех значений:

```
int myMaxValue = max(analogRead(0),
                     max(analogRead(1),
                         max(analogRead(2), analogRead(3))));
```

Дополнительная информация

См. также *разд. 3.5*.

3.7. Возведение числа в степень

ЗАДАЧА

Требуется возвести число в степень.

РЕШЕНИЕ

Функция `pow(x, y)` возвращает значение переменной `x`, возведенное в степень значения переменной `y`:

```
int myValue = pow(3,2);
```

В этом примере вычисляется возведение числа 3 в степень 2, в результате чего значению переменной `myValue` присваивается значение 9.

Обсуждение работы решения и возможных проблем

Функция `pow()` может работать как с целыми числами, так и с числами с плавающей запятой, но возвращает результат в числе с плавающей запятой:

```
Serial.println(pow(3,2)); // Выводится значение 9.00
int z = pow(3,2);
Serial.println(z);        // Выводится значение 9
```

В первом случае функция `println()` выводит значение 9.00, а во втором — 9. Это не совсем одинаковые значения, поскольку первое является числом с плавающей запятой (точкой), а второе — целочисленным значением типа `int`, что и объясняет отсутствие десятичной точки. Более подробно различие между этими двумя типами данных рассматривается в *разд. 2.3*.

В следующем примере число возводится в дробную степень:

```
float s = pow(2, 1.0 / 12); // двенадцатый корень из числа 2
```

Двенадцатый корень из двух — то же самое, что 2 в степени 0,083333. В обоих случаях результат будет 1,05946 (это соотношение частот двух смежных нот пианино).

3.8. Извлечение квадратного корня

ЗАДАЧА

Требуется вычислить квадратный корень числа.

РЕШЕНИЕ

Функция `sqrt(x)` возвращает квадратный корень значения параметра `x`:

```
Serial.println( sqrt(9) ); // Выводится значение 3.00
```

Обсуждение работы решения и возможных проблем

Функция `sqrt()` возвращает результат в виде числа с плавающей запятой (см. обсуждение функции `pow()` в *разд. 3.7*).

3.9. Округление с повышением и понижением чисел с плавающей запятой

ЗАДАЧА

Требуется получить ближайшее наименьшее или наибольшее целое значение числа с плавающей запятой.

РЕШЕНИЕ

Функция `floor(x)` возвращает наибольшее целое значение, не превышающее значение параметра `x`, а функция `ceil(x)` возвращает наименьшее целое число, не меньшее, чем значение параметра `x`.

Обсуждение работы решения и возможных проблем

Эти функции можно использовать для округления чисел с плавающей запятой (точкой). Функция `floor()` возвращает наибольшее целое число, не большее, чем значение параметра `x`, а функция `ceil()` возвращает наименьшее целое число, большее, чем значение параметра `x`.

В листинге 3.9 приводится несколько примеров использования функции `floor()`.

Листинг 3.9. Примеры использования функции `floor()`

```
Serial.println( floor(1) );      // Выводится значение 1.00
Serial.println( floor(1.1) );    // Выводится значение 1.00
Serial.println( floor(0) );      // Выводится значение 0.00
Serial.println( floor(.1) );     // Выводится значение 0.00
Serial.println( floor(-1) );     // Выводится значение -1.00
Serial.println( floor(-1.1) );   // Выводится значение -2.00
```

А в листинге 3.10 приводится несколько примеров использования функции `ceil()`.

Листинг 3.10. Примеры использования функции `ceil()`

```
Serial.println( ceil(1) );       // Выводится значение 1.00
Serial.println( ceil(1.1) );     // Выводится значение 2.00
Serial.println( ceil(0) );       // Выводится значение 0.00
Serial.println( ceil(.1) );      // Выводится значение 1.00
Serial.println( ceil(-1) );      // Выводится значение -1.00
Serial.println( ceil(-1.1) );    // Выводится значение -1.00
```

Числа с плавающей запятой (точкой) можно округлять до ближайшего целого числа следующим образом:

```
int result = round(1.1);
```

Числа с плавающей запятой можно усекать, преобразовывая их тип данных в `int`, но округление посредством этом метода выполняется неправильно. В частности, отрицательное число, например –1,9, должно округляться с уменьшением до –2, но при преобразовании в целое число оно округляется до –1. Такая же проблема наблюдается и для положительных чисел. Например, число 1,9 должно округлиться с повышением до 2, но округляется с понижением до 1. Чтобы получить правильные результаты, следует использовать функции `floor()`, `ceil()` и `round()`.

3.10. Работа с тригонометрическими функциями

ЗАДАЧА

Требуется вычислить синус, косинус и тангенс угла, заданного в радианах или градусах.

РЕШЕНИЕ

Функции `sin(x)`, `cos(x)` и `tan(x)` возвращают синус, косинус и тангенс угла, заданного в параметре `x` соответственно.

Обсуждение решения и возможных проблем

Значения углов в параметре `x` функций указываются в радианах, и функции возвращают результат в виде числа с плавающей запятой (см. *разд. 2.3*). В листинге 3.11 приводится несколько примеров использования тригонометрических функций.

Листинг 3.11. Примеры использования тригонометрических функций

```
float deg = 30;                // Значение угла в градусах
float rad = deg * PI / 180;    // Преобразовываем в радианы
Serial.println(rad);           // Выводим значение угла в радианах
Serial.println(sin(rad), 5);   // Выводим значение синуса
Serial.println(cos(rad), 5);   // Выводим значение косинуса
```

Код этого примера преобразовывает значение угла, заданное в градусах, в соответствующее значение в радианах, а затем вычисляет и выводит на экран значения синуса и косинуса этого угла. Далее приводится пример вывода этого кода с добавленными пояснениями:

```
0.52   30 градусов равно 0.5235988 радиан, но println отображает
    // только два десятичных знака
0.50000 синус угла 30 градусов равен .5000000, но здесь он
    // отображен с точностью до 5 знаков после запятой (точки)
0.86603 косинус равен .8660254, что при округлении до 5 знаков
    // после запятой отображается как 0.86603
```

Хотя приведенный код вычисляет эти значения до полной точности чисел с плавающей запятой, функции `Serial.print()` и `Serial.println()` по умолчанию отображают вычисленные значения с точностью только до двух десятичных знаков. Но требуемую точность вывода этими функциями можно указать в их втором параметре, как показано в *разд. 2.3*. И в нашем примере для вывода значений синуса и косинуса была указана точность до 5 знаков после запятой (точки).

Преобразование значения угла из градусов в радианы и обратно является стандартной задачей учебника по тригонометрии. Идентификатор `PI` представляет стандартную константу π (3,14159265...). Идентификатор `PI` и значение 180 являются константами, но среда Arduino IDE также предоставляет пару предварительно вычисленных констант для преобразований между градусами и радианами:

```
rad = deg * DEG_TO_RAD; // Преобразование градусов в радианы
deg = rad * RAD_TO_DEG; // Преобразование радиан в градусы
```

Может показаться, что использование выражения: `deg * DEG_TO_RAD` будет более эффективным, чем выражения: `deg * PI / 180`, но это не так. Компилятор Arduino достаточно умный, чтобы сообразить, что выражение `PI / 180` является константой (т. е. значением, которое никогда не меняется), поэтому он заменяет это выражение его вычисленным значением, которое оказывается таким же, как и значение константы `DEG_TO_RAD` (0.017453292519...). Поэтому можно использовать любое из этих выражений, которое вам нравится больше.

Дополнительная информация

Дополнительная информация по функциям `sin()`, `cos()` и `tan()` приводится на страницах справки веб-сайта Arduino: **https://oreil.ly/7RvuR**, **https://oreil.ly/iPiVA** и **https://oreil.ly/Xm-lN** соответственно.

3.11. Генерация произвольных чисел

ЗАДАЧА

Требуется генерировать произвольное число в диапазоне значений от 0 до определенного максимального значения или в пределах указанных минимального и максимального значений.

РЕШЕНИЕ

Эта задача решается посредством использования функции `random()`. При вызове функции `random()` с одним параметром задается верхний предел, и она возвратит число в диапазоне от нуля до заданного предела (не включая сам предел), как показано в следующем примере:

```
int minr = 50;
int maxr = 100;
long randnum = random(maxr); // Произвольное число в диапазоне от 0 до (maxr-1)
```

При вызове функции `random()` с двумя параметрами задаются нижний и верхний пределы, и она возвратит число в диапазоне от нижнего предела (включая само значение предела) до верхнего предела (не включая сам предел), как показано в следующем примере:

```
long randnum = random(minr, maxr); // Произвольное число в диапазоне
                                   // от minr до (maxr-1)
```

Обсуждение работы решения и возможных проблем

Хотя на первый взгляд кажется, что функция возвращает действительно произвольные числа, на самом деле это не так. При каждом запуске скетча будет генерироваться одна и та же последовательность значений. Для многих приложений это обстоятельство не имеет большого значения. Но если необходимо, чтобы при каждом запуске скетча генерировалась другая последовательность чисел, нужно использовать функцию `randomSeed(`*начальное_значение*`)` с разным начальным значением при каждом запуске (если использовать одно и то же начальное значение, получим одну и ту же последовательность.) Эта функция запускает генератор случайных чисел в произвольном месте последовательности — в зависимости от начального значения, передаваемого функции `randomSeed()`:

```
randomSeed(1234); // Меняем начало последовательности случайных чисел
```

В листинге 3.12 демонстрируется использование нескольких разных типов генерации случайных чисел, доступных в языке Arduino:

Листинг 3.12. Несколько способов генерации случайных чисел

```
// Скетч Random
// Демонстрирует генерирование произвольных чисел

int randNumber;

void setup()
{
   Serial.begin(9600);
   while(!Serial);

   // Генерируем последовательность произвольных чисел,
   // не используя начальное значение, и выводим в окно монитора порта
   Serial.println("Print 20 random numbers between 0 and 9");
        // Выводим на экран 20 произвольных чисел в диапазоне от 0 до 9
   for(int i=0; i < 20; i++)
   {
      randNumber = random(10);
      Serial.print(randNumber);
      Serial.print(" ");
   }
```

```
  Serial.println();
  Serial.println("Print 20 random numbers between 2 and 9");
       // Генерируем 20 произвольных чисел в диапазоне от 2 до 9
  for(int i=0; i < 20; i++)
  {
     randNumber = random(2,10);
     Serial.print(randNumber);
     Serial.print(" ");
  }

  // Генерируем последовательность произвольных чисел,
  // используя каждый раз одинаковое начальное значение,
  // и выводим в окно монитора порта
  randomSeed(1234);
  Serial.println();
  Serial.println("Print 20 random numbers between 0 and 9 after constant seed ");
       // Генерируем 20 произвольных чисел в диапазоне от 0 до 9,
       // используя одинаковое начальное значение
  for(int i=0; i < 20; i++)
  {
     randNumber = random(10);
     Serial.print(randNumber);
     Serial.print(" ");
  }

  // Генерируем последовательность произвольных чисел, используя
  // каждый раз другое начальное значение, и выводим в окно монитора порта
  randomSeed(analogRead(0)); // Считываем входное значение аналогового
                             // контакта, к которому ничего не подключено
  Serial.println();
  Serial.println("Print 20 random numbers between 0 and 9 after floating seed ");
       // Генерируем 20 произвольных чисел в диапазоне от 0 до 9,
       // используя разные начальные значения
  for(int i=0; i < 20; i++)
  {
     randNumber = random(10);
     Serial.print(randNumber);
     Serial.print(" ");
  }
  Serial.println();
  Serial.println();
}

void loop()
{
}
```

Результаты исполнения этого скетча на плате Arduino Uno выглядят следующим образом (для плат с другой архитектурой результат может быть иным):

```
Print 20 random numbers between 0 and 9
(Выводим на экран 20 произвольных чисел в диапазоне от 0 до 9)
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0 2
Print 20 random numbers between 2 and 9
(Выводим на экран 20 произвольных чисел в диапазоне от 2 до 9)
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
Print 20 random numbers between 0 and 9 after constant seed
(Генерируем 20 произвольных чисел в диапазоне от 0 до 9,
 используя одинаковое начальное значение)
8 2 8 7 1 8 0 3 6 5 9 0 3 4 3 1 2 3 9 4
Print 20 random numbers between 0 and 9 after floating seed
(Генерируем 20 произвольных чисел в диапазоне от 0 до 9,
 используя разные начальные значения)
0 9 7 4 4 7 7 4 4 9 1 6 0 2 3 1 5 9 1 1
```

Если перезапускать исполнение скетча, нажимая кнопку сброса платы Arduino, то первые три набора произвольных чисел будут такими же при каждом исполнении скетча (после сброса может потребоваться закрыть и снова запустить монитор порта). Только последний набор произвольных чисел будет разным при каждом новом исполнении скетча, поскольку при каждом новом запуске скетча используется разное начальное значение для функции `randomSeed()`, получаемое чтением входного сигнала с неподключенного аналогового контакта. Если аналоговый контакт используется для каких-либо других целей, следует изменить этот параметр функции `analogRead()` на номер неиспользуемого аналогового контакта.

В общем, рассмотренный пример исчерпывает возможности генерации произвольных чисел на плате Arduino, не прибегая к использованию внешних аппаратных устройств. На первый взгляд, использование входного значения неподключенного аналогового контакта может казаться хорошим или, по крайней мере, приемлемым способом создания начального значения для генератора случайных чисел. Но аналого-цифровой преобразователь большинства плат Arduino может возвращать самое большее 10-разрядное двоичное число, которое может представлять всего лишь 1024 разных значений. Это слишком малый диапазон начальных значений для генерации сильных случайных чисел. Кроме того, значение, создаваемое неподключенным аналоговым контактом, не такое и случайное, каким, можно подумать, оно должно быть. Оно может проявлять в некоторой степени постоянные закономерности, и кто угодно, оказавшийся вблизи платы Arduino, может оказывать на него влияние.

Генерация по-настоящему случайных чисел на плате Arduino — задача не из легких. Но, как и в случае с большинством компьютеров, с ее помощью можно генерировать криптостойкие псевдослучайные числа, т. е. числа, достаточно случайные, чтобы их можно было использовать в криптографических приложениях. Некоторые платы Arduino — например, Arduino WiFi Rev2, MKR Vidor 4000 и MKR WiFi 1000/1010 — оснащены криптографическим сопроцессором ECC508 или ECC608

компании Atmel, аппаратно реализующим криптографические функции, включая генерацию сильных случайных чисел. Для работы с этим сопроцессором необходимо с помощью менеджера библиотек среды Arduino IDE установить библиотеку ArduinoECCX08 (подробная информация по установке библиотек приводится в *разд. 16.2*). Для генерации сильных случайных чисел на любой плате Arduino можно также использовать библиотеку Crypto (**https://oreil.ly/80KEp**) разработки Риса Уэзерли (Rhys Weatherley) — в частности, класс RNG (**https://oreil.ly/33djZ**).

Дополнительная информация

Дополнительная информация по функциям `random()` и `randomSeed()` приводится на страницах справки веб-сайта Arduino: **https://oreil.ly/lDIp7** и **https://oreil.ly/zYbaF** соответственно.

3.12. Установка и считывание значений битов

ЗАДАЧА

Требуется считать или задать значение определенного бита числовой переменной.

РЕШЕНИЕ

В зависимости от конкретного типа задачи используется одна из следующих функций:

- `bitSet(x, bitPosition)`

  Устанавливает (записывает значение 1) бит в указанной позиции `bitPosition` переменной `x`.

- `bitClear(x, bitPosition)`

  Сбрасывает (записывает значение 0) бит в указанной позиции `bitPosition` переменной `x`.

- `bitRead(x, bitPosition)`

  Возвращает значение (0 или 1) бита в указанной позиции `bitPosition` переменной `x`.

- `bitWrite(x, bitPosition, value)`

  Устанавливает значение (0 или 1) бита в указанной позиции `bitPosition` переменной `x`.

- `bit(bitPosition)`

  Возвращает значение позиции бита: `bit(0)` равняется 1, `bit(1)` — 2, `bit(2)` — 4 и т. д.

Во всех этих функциях позиция `bitPosition` 0 является самым младшим (крайним правым) битом.

В листинге 3.13 приводится скетч, использующий все эти функции для манипуляции битами 8-разрядной переменной с названием `flags`. Каждый из битов этой

переменной представляет отдельный флаг, который можно устанавливать или сбрасывать.

Листинг 3.13. Демонстрация манипулирования отдельными битами 8-разрядного числа

```
// Скетч bitFunctions
// Демонстрирует работу функций манипулирования битами

byte flags = 0;

// Эти примеры устанавливают, сбрасывают или считывают биты
// в переменной с названием flags

// Пример функции bitSet()

void setFlag(int flagNumber)
{
   bitSet(flags, flagNumber);
}

// Пример функции bitClear()
void clearFlag(int flagNumber)
{
   bitClear(flags, flagNumber);
}

// Пример функции bitPosition()
int getFlag(int flagNumber)
{
   return bitRead(flags, flagNumber);
}

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

void loop()
{
   flags = 0;      // Обнуляем все флаги
   showFlags();
   setFlag(2);     // Устанавливаем некоторые флаги
   setFlag(5);
   showFlags();
   clearFlag(2);
   showFlags();
```

```
   delay(10000); // Очень длинная пауза
}

// Докладываем, какие флаги установлены
void showFlags()
{
   for(int flag=0; flag < 8; flag++)
   {
      if (getFlag(flag) == true)
      Serial.print("* bit set for flag "); else
      Serial.print("bit clear for flag ");

      Serial.println(flag);
   }
   Serial.println();
}
```

Каждые 10 секунд код будет выводить в окне монитора порта следующее:

```
bit clear for flag 0
bit clear for flag 1
bit clear for flag 2
bit clear for flag 3
bit clear for flag 4
bit clear for flag 5
bit clear for flag 6
bit clear for flag 7

bit clear for flag 0
bit clear for flag 1
* bit set for flag 2
bit clear for flag 3
bit clear for flag 4
* bit set for flag 5
bit clear for flag 6
bit clear for flag 7

bit clear for flag 0
bit clear for flag 1
bit clear for flag 2
bit clear for flag 3
bit clear for flag 4
* bit set for flag 5
bit clear for flag 6
bit clear for flag 7
```

Обсуждение работы решения и возможных проблем

Считывание и установка состояния битов являются повсеместной задачей, и многие библиотеки Arduino используют эту функциональность. Одно из распространенных применений операций манипулирования битами — эффективно сохранять и извлекать двоичные значения (включено/выключено, истина/ложь, 1/0, высокий/низкий и т. п.).

В языке Arduino для констант `true` (истина) и `HIGH` (высокий уровень) определено значение 1, а для `false` (ложь) и `LOW` (низкий уровень) — 0.

Состояния восьми переключателей — вместо восьми значений типа `byte` или `int` — можно упаковать в одно 8-битовое значение. В примере решения этого раздела (см. листинг 3.13) демонстрируется, как можно задавать значения отдельным битам байта.

В области программирования термин *флаг* применяется к значениям, которые хранят какой-либо аспект программы. В приведенном скетче биты флагов считываются посредством функции `bitRead()` и устанавливаются или сбрасываются с помощью функций `bitSet()` и `bitClear()` соответственно. Этим функциям передаются два параметра: первый содержит значение (переменная `flags` в нашем примере), содержащее бит для чтения или записи, а второй — позицию требуемого бита в этом значении. Позиция самого младшего (крайнего правого) бита — 0, следующего — 1, и т. д., до самого старшего (крайнего левого) бита с позицией 7. Таким образом:

```
bitRead(2, 1); // Возвращает 1 : десятичное 2 равно двоичному 10,
               // а значение бита в позиции 1 равно 1
bitRead(4, 1); // Возвращает 0 : десятичное 4 равно двоичному 100,
               // а значение бита в позиции 1 равно 0
```

Функция `bit()` возвращает десятичный вес каждой битовой позиции:

```
bit(0) возвращает 1;
bit(1) возвращает 2;
bit(2) возвращает 4;
...
bit(7) возвращает 128
```

Дополнительная информация

Дополнительная информация по рассмотренным функциям приводится на странице справки веб-сайта Arduino:

- `lowByte` (**https://oreil.ly/aHPmW**);
- `highByte` (**https://oreil.ly/9M2Ob**);
- `bitRead` (**https://oreil.ly/gGWGT**);
- `bitWrite` (**https://oreil.ly/H5NO0**);

- `bitSet` (**https://oreil.ly/GTzsQ**);
- `bitClear` (**https://oreil.ly/3waji**);
- `bit` (**https://oreil.ly/YPNrk**).

3.13. Смещение битов

ЗАДАЧА

Требуется выполнить операцию над битами двоичного числа типа `byte`, `int` или `long`, состоящую в смещении его битов вправо или влево.

РЕШЕНИЕ

Составляющие биты двоичного числа можно сместить влево посредством оператора <<, а вправо — посредством оператора >>.

Обсуждение работы решения и возможных проблем

Фрагмент кода в листинге 3.14 задает переменной `x` значение 6, смещает биты переменной на одну позицию влево и выводит в монитор порта новое значение переменной (12). Затем биты этого нового значения смещаются на две позиции вправо, и полученное значение (3) снова выводится в окно монитора порта.

Листинг 3.14. Пример операций смещения битов

```
int x = 6;
x = x << 1; // Смещение битов числа 6 на одну позицию влево дает число 12
Serial.println(x);
x = x >> 2; // Смещение битов числа 12 на две позиции вправо дает число 3
Serial.println(x);
```

Двоичное значение десятичного числа 6 равно 0110, или 0 + 4 + 2 + 0. Смещение битов на одну позицию влево дает двоичное 1100, что равно десятичному 12, поскольку 8 + 4 + 0 + 0 = 12. А смещение битов этого нового числа на две позиции вправо дает двоичное 0011, что равно десятичному числу 3, поскольку 0 + 0 + 2 + 1 = = 3. Обратите внимание: смещение битов двоичного числа влево на *n* позиций равнозначно умножению исходного значения на 2^n (2 в степени *n*). А смещение битов двоичного числа вправо на *n* позиций равнозначно делению исходного значения на 2^n (2 в степени *n*). Иными словами, следующие пары выражений дают одинаковый результат:

`x << 1` равнозначно `x * 2`

`x << 2` равнозначно `x * 4`

`x << 3` равнозначно `x * 8`

`x >> 1` равнозначно `x / 2`

`x >> 2` равнозначно `x / 4`

`x >> 3` равнозначно `x / 8`

Микроконтроллер платы Arduino выполняет операции смещения битов более эффективно, чем операции умножения и деления, поэтому операции смещения битов могут использоваться для выполнения операций умножения и деления:

```
int c = (a << 1) + (b >> 2); //(a умноженное на 2) плюс (b разделенное на 4)
```

Выражение

```
(a << 1) + (b >> 2);
```

не очень похоже на выражение

```
(a * 2) + (b / 4);
```

но оба эти выражения дают одинаковый результат. Более того, компилятор Arduino достаточно умный, чтобы видеть, что умножение целого числа на константу, равную степени двух, равнозначно смещению битов числа влево на количество позиций, равное величине степени, и даст такой же результат, как и операция смещения. Но исходный код с использованием арифметических операторов более удобен для понимания пользователями, поэтому лучше использовать эти операторы, когда целью является именно умножение или деление.

Дополнительная информация

Дополнительная информация по функциям `lowByte()`, `highByte()`, `bitRead()`, `bitWrite()`, `bitSet()`, `bitClear()` и `bit()` приводится на страницах справки веб-сайта Arduino (см. соответствующие ссылки в *разд. 3.12*).

3.14. Извлечение старшего или младшего байта из значений типа *int* и *long*

ЗАДАЧА

Требуется извлечь старший или младший байт целочисленного значения. Например, для передачи целочисленных значений в виде байтов в последовательном обмене данными.

РЕШЕНИЕ

Младший байт целочисленного значения можно извлечь с помощью функции `lowByte()`, а старший — посредством функции `highByte()`.

В листинге 3.15 приводится пример скетча, который преобразовывает целочисленное значение в младший и старший байты.

Листинг 3.15. Извлечение младшего и старшего байтов из целого числа

```
/*
* Скетч ByteOperators
*/

int intValue = 258; // Десятичное 258 в шестнадцатеричном формате равно 0x102

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

void loop()
{
   int loWord,hiWord;
   byte loByte, hiByte;

   hiByte = highByte(intValue);
   loByte = lowByte(intValue);

   Serial.println(intValue, DEC);
   Serial.println(intValue, HEX);
   Serial.println(loByte, DEC);
   Serial.println(hiByte, DEC);

   delay(10000); // Очень длинная пауза
}
```

Обсуждение работы решения и возможных проблем

Скетч этого примера выводит в окно монитора порта значение переменной `intValue` в десятичном и шестнадцатеричном формате, а затем младший и старший байты этого значения:

```
258     (исходное целочисленное значение)
102     (исходное значение в шестнадцатеричном формате)
2       (младший байт)
1       (старший байт)
```

Чтобы извлечь байт из целого длинного значения, это 32-битовое значение нужно сначала разбить на два 16-битовых слова, а уже из этих слов извлекаются младший и старший байты, как показано в листинге 3.16. На момент подготовки этой книги стандартная библиотека Arduino не содержит функции для реализации такого разбиения, но его можно выполнить посредством следующих двух строк кода:

```
#define highWord(w) ((w) >> 16)
#define lowWord(w) ((w) & 0xffff)
```

Макрос `highWord()` выполняет операцию смещения битов значения вправо на 16 позиций, преобразовывая его в 16-битовое значение, а макрос `lowWord()` накладывает маску на младшие 16 битов значения с помощью побитового оператора и (`&`) (см. *разд. 2.20*).

Количество битов целого числа типа `int` зависит от платформы программирования. В среде разработки Arduino IDE это 16 битов, но в других средах — 32 бита. Термин «слово» в данном контексте означает 16-разрядное значение.

Код в листинге 3.16 преобразовывает 32-битовое значение 16909060 (0x1020304 в шестнадцатеричном формате) в его составляющие 16-разрядные части.

Листинг 3.16. Преобразование 32-разрядного значения в составляющие 16-разрядные значения

```
long longValue = 16909060;
int loWord = lowWord(longValue);
int hiWord = highWord(longValue);
Serial.println(loWord, DEC);
Serial.println(hiWord, DEC);
```

Этот скетч выводит в окно монитора порта следующие значения (без добавленных здесь пояснений):

```
772 (772 в шестнадцатеричном формате будет 0x0304)
258 (258 в шестнадцатеричном формате будет 0x0102)
```

Обратите внимание: десятеричное значение 772 в шестнадцатеричном формате будет 0x0304, а это младшее слово (16-бит) значения 0x1020304 переменной `longValue`. А десятичное значение 258 является результатом комбинирования старшего байта со значением 1 и младшего со значением 2 (или 0x0102 в шестнадцатеричном формате).

Дополнительная информация

Дополнительная информация по функциям `lowByte()`, `highByte()`, `bitRead()`, `bitWrite()`, `bitSet()`, `bitClear()` и `bit()` приводится на страницах справки веб-сайта Arduino (см. соответствующие ссылки в *разд. 3.12*).

3.15. Создание целого числа типа *int* или *long* из отдельных байтов

ЗАДАЧА

Требуется создать 16-разрядное (типа `int`) или 32-разрядное (типа `long`) целочисленное значение из отдельных байтов — например, при приеме значений в виде

отдельных байтов по последовательному каналу связи. Это обратная операция решения из *разд. 3.14*.

РЕШЕНИЕ

Сложить два отдельных байта в целое число можно с помощью функции `word(h,l)`. Соответствующий код приводится в листинге 3.17. Это, по сути, код из листинга 3.16, дополненный кодом для преобразования извлеченного старшего и младшего байтов обратно в одно число.

Листинг 3.17. Создание целого значения типа `int` из отдельных байтов

```
/*
* Создание целого числа из отдельных байтов
*/

int intValue = 0x102; // 258

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

void loop()
{
   int aWord;
   byte loByte, hiByte;

   hiByte = highByte(intValue);
   loByte = lowByte(intValue);

   Serial.println(intValue, DEC);
   Serial.println(loByte, DEC);
   Serial.println(hiByte, DEC);

   aWord = word(hiByte, loByte); // Преобразовываем извлеченные байты
                                 // снова в одно слово
   Serial.println(aWord, DEC);
   delay(10000); // Очень длинная пауза
}
```

Обсуждение работы решения и возможных проблем

Оператор `word(high,low)` собирает 16-битное значение из старшего и младшего байтов. Код в листинге 3.18 сначала извлекает старший и младший байты из целого числа, а затем собирает их обратно в одно слово. Скетч выводит в окно монитора порта результаты своей работы: исходное целое значение (типа `int`), младший байт, старший байт и собранное снова из этих байт целое число:

```
258
2
1
258
```

На момент подготовки этой книги язык Arduino не содержал функции для преобразования целого числа типа `long` в два 16-разрядные слова. Но этот код можно оснастить такой функциональностью, добавив в начале макрос `makeLong()`:

```
#define makeLong(hi, low) ((hi) << 16 & (low))
```

Макрос смещает старшее значение на 16 позиций влево и складывает его с младшим значением. Этот макрос используется в следующем скетче (листинг 3.18), где целое число типа `long` с помощью макросов `highWord()` и `lowWord()` разбивается на составляющие 16-разрядные значения, а затем из этих значений складывается снова в целое число типа `long`.

Листинг 3.18. Создание целого значения типа `long` из отдельных 16-разрядных чисел

```
#define makeLong(hi, low) (((long) hi) << 16 | (low))
#define highWord(w) ((w) >> 16)
#define lowWord(w) ((w) & 0xffff)
// Объявляем исходное значение
long longValue = 0x1020304; // В десятичном формате: 16909060
// В двоичном формате: 00000001 00000010 00000011 00000100
void setup()
{
   Serial.begin(9600);
}

void loop()
{
   int loWord,hiWord;
   Serial.println(longValue,DEC); // Выводим в монитор порта
                                  // значение 16909060
   loWord = lowWord(longValue);   // Преобразовываем исходное значение в два слова
   hiWord = highWord(longValue);
   Serial.println(loWord,DEC);    // Выводим в монитор порта
                                  // младшее значение (слово) 772
   Serial.println(hiWord,DEC);    // Выводим в монитор порта
                                  // старшее значение (слово) 258
   longValue = makeLong(hiWord, loWord); // Преобразовываем извлеченные
                                         // слова обратно в одно значение long
   Serial.println(longValue,DEC);  // Снова выводим в монитор порта
                                   // значение 16909060
   delay(10000);              // Очень длинная пауза
}
```

Скетч выводит в окно монитора порта результаты своей работы: исходное целое значение (типа `long`), младшее слово, старшее слово и собранное снова из этих слов целое число типа `long`:

```
16909060
772
258
16909060
```

Термин «слово» означает 16-разрядное значение в контексте кода для 8-разрядных плат — например, Arduino Uno. В контексте кода для 32-разрядных плат «слово» означает 32-разрядное значение, а «полуслово» — 16-разрядное. Но хотя архитектура плат может быть разной, код в листинге 3.19 возвратит старшее и младшее 16-разрядные значения как для 8-разрядных плат, так и для 32-разрядных.

Дополнительная информация

Дополнительная информация по функциям `lowByte()`, `highByte()`, `bitRead()`, `bitWrite()`, `bitSet()`, `bitClear()` и `bit()` приводится на страницах справки веб-сайта Arduino (см. соответствующие ссылки в *разд. 3.12*).

ГЛАВА 4

Последовательная связь

4.0. Введение

Последовательная связь предоставляет легкий и гибкий способ для взаимодействия платы Arduino с компьютером или другими устройствами. В этой главе мы рассмотрим, как с помощью последовательной связи осуществлять обмен информацией.

Из *главы 1* мы узнали, как использовать последовательный порт USB платы Arduino для загрузки в нее скетчей. В процессе загрузки данные отправляются с компьютера на плату Arduino, которая, в свою очередь, отправляет на компьютер сообщения, подтверждающие нормальный ход этого процесса. Примеры, приведенные в главе, которую мы начинаем сейчас изучать, показывают возможности использования последовательного канала связи для обмена любой информацией между платой Arduino и компьютером или каким-либо другим устройством, оснащенным такой возможностью.

Последовательная связь также хорошо подходит в качестве удобного инструмента для диагностирования неполадок и отладки приложений. Отладочные сообщения можно отправлять с платы Arduino на компьютер, отображая их на его экране, или на другие устройства — например, на одноплатный компьютер Raspberry Pi или другую плату Arduino. Эти сообщения можно также отображать на внешнем жидкокристаллическом дисплее, но взаимодействие с таким дисплеем, скорее всего, будет лучше осуществлять посредством интерфейса связи I^2C или SPI (см. *главу 13*).

Среда разработки Arduino IDE, которую мы рассмотрели в *разд. 1.3*, содержит инструмент *Монитор порта* (рис. 4.1), предназначенный для обмена данными с платой Arduino по каналу последовательной связи. В частности, с помощью монитора порта можно отправлять данные на плату Arduino, вводя их в текстовую строку, расположенную в верхней части окна монитора порта, и нажимая кнопку **Отправить** справа от этой строки. А принимаемые от платы Arduino данные отображаются в главной панели монитора порта. Среда Arduino IDE также оснащена инструментом *Плоттер по последовательному порту*, способным создавать графики данных, получаемых от платы Arduino по последовательному каналу связи. Мы рассмотрим использование этого инструмента в *разд. 4.1*.

Скорость обмена данными (в бодах, или битах в секунду) устанавливается выбором требуемого значения из выпадающего списка, открываемого нажатием на кнопку **9600 бод** в правой нижней части окна монитора порта (на этой кнопке может быть

отображена и другая надпись, соответствующая выбранной ранее скорости обмена). Выбранное значение должно соответствовать скорости обмена, заданной в скетче посредством функции `Serial.begin()`. В большинстве случаев обычно используется скорость обмена величиной 9600 бод, но при работе с устройством, требующим более высокой скорости, в функции `Serial.begin()` можно задать и другую необходимую скорость.

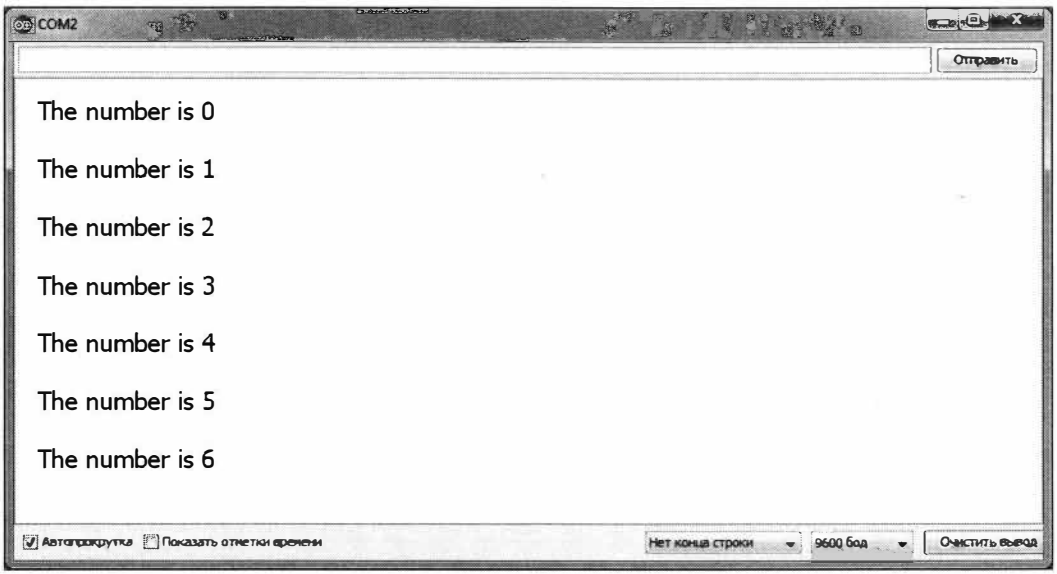


Рис. 4.1. Окно инструмента **Монитор порта** среды Arduino IDE

Кроме скорости обмена в мониторе порта в выпадающем списке, открываемом нажатием на кнопку **Нет конца строки** в правой нижней части окна монитора порта (на этой кнопке может быть отображена и другая надпись, соответствующая выбранной опции), можно задать способ завершения передаваемого на плату Arduino сообщения. Доступны четыре опции: **NL (Новая строка)** (значение ASCII 10), **CR (Возврат каретки)** (значение ASCII 13), **NL&CR** (комбинация символов новой строки и возврата каретки) или же отсутствие какого-либо разделителя **Нет конца строки**.

С помощью последовательного канала связи скетч Arduino может косвенно (обычно используя программу-посредник на языке Processing или Python) обращаться ко всем ресурсам компьютера, к которому подключена плата: памяти, экрану, клавиатуре, мыши, сетевому подключению и т. п.). Тем же самым образом посредством канала последовательной связи компьютер может взаимодействовать с определенными датчиками или иными устройствами, подключенными к плате Arduino. Чтобы по последовательному каналу связи компьютер мог одновременно взаимодействовать с несколькими устройствами, необходимо или оснастить компьютер дополнительными последовательными портами, или же задействовать программную последовательную связь для эмулирования последовательного порта, используя контакты платы Arduino. Вторая опция рассматривается более подробно далее,

в *разд.* «*Эмуляция аппаратной последовательной связи посредством цифровых контактов ввода/вывода*».

Многие датчики и устройства вывода, поддерживающие обычную последовательную связь, также поддерживают связь по протоколу SPI или I^2C (эти протоколы рассматриваются более подробно в *главе 13*). Тогда как обычная последовательная связь широко известна и в некоторой степени универсальна, возможность использования связи по протоколу SPI или I^2C следует предусматривать только в том случае, когда один (или оба) из этих протоколов поддерживается датчиком или устройством вывода, с которым необходимо организовать обмен данными. Оба протокола предоставляют большую гибкость при взаимодействии с множественными устройствами.

Для реализации последовательной связи необходимо задействовать как аппаратные, так и программные средства. Аппаратные средства обеспечивают обмен электрическими сигналами между платой Arduino и подключенным к ней устройством, а программные средства используют аппаратные для обмена байтами или битами, которые понимает аппаратная часть. Библиотеки среды Arduino IDE скрывают большую часть низкоуровневых подробностей аппаратных средств от пользователя, но весьма полезно понимать основы, особенно если вам придется диагностировать какие-либо проблемы с последовательной связью в своих проектах.

Аппаратные средства последовательной связи

Аппаратные средства последовательной связи обеспечивают обмен данными в виде электрических импульсов, представляющих последовательные биты. Нули и единицы байта информации можно представить разными способами. В используемой средой Arduino IDE схеме напряжение величиной 0 В представляет логическое значение 0, а величиной 5 В (или 3,3 В) — логическое значение 1.

Схема, в которой низкое напряжение устройства (обычно 0 В) представляет логический 0, а высокое (3,3 или 5 В в случае Arduino) — логическую 1, имеет широкое распространение. Такой способ представления логических уровней называется *ТТЛ-логикой*, поскольку так представлялись логические значения в одной из первых реализаций цифровой логики, называемой *транзисторно-транзисторной логикой* (ТТЛ). В большинстве реализаций ТТЛ-логики логическое значение 1 можно представлять сигналом, напряжение которого меньше, чем максимальное положительное напряжение устройства, и обычно напряжения величиной 3,3 В более чем достаточно для представления логического значения 1. Это означает, что в большинстве случаев сигнал, передаваемый платой с рабочим напряжением 3,3 В, можно принимать на плате с рабочим напряжением 5 В. Но обратная передача, т. е. с платы с рабочим напряжением 5 В на плату с рабочим напряжением 3,3 В, в большинстве случаев невозможна. Чтобы реализовать такую передачу, нужно использовать схему сдвига уровня или делитель напряжения, позволяющий избежать повреждения платы с рабочим напряжением 3,3 В. Делители напряжения рассматриваются в *разд. 4.13* и *5.11*.

Некоторые платы, например платы Bare Bones Board компании Modern Device, Boardino компании Adafruit (в настоящее время снята с производства) и Arduino Pro, Mini и Pro Mini, не оснащены поддержкой USB, и для подключения этих плат к компьютеру требуется специальный адаптер, преобразовывающий сигналы ТТЛ

в сигналы USB. В качестве такого адаптера с одинаковым успехом можно использовать адаптеры CP2104 Friend компании Adafruit (артикул 3309), USB BUB компании Modern Device (артикул MD022X) и USB-TTL Adapter компании FTDI (**https://oreil.ly/2FhMP**).

Некоторые устройства с поддержкой последовательной связи используют для подключения стандарт RS-232. Обычно они оснащены 9-контактным разъемом и для подключения к плате Arduino требуют специального адаптера. Используемые в последовательном интерфейсе RS-232 уровни напряжения способны вывести из строя цифровые контакты платы Arduino, поэтому для работы с ним необходимо применять адаптер RS-232/ТТЛ. На веб-сайте Arduino предлагается руководство по работе с интерфейсом RS-232 (**https://oreil.ly/IVXKQ**), а на веб-сайте Serial Port Central (**https://oreil.ly/cB1Jr**) — большой объем информации и ссылок по этому интерфейсу.

Плата Arduino Uno оснащена одним аппаратным последовательным портом, но в ней можно также эмулировать дополнительные последовательные порты (каналы связи) с помощью программных библиотек. Таким образом можно обеспечить возможность последовательной связи для нескольких подключенных к плате устройств. Для реализации последовательной связи программными средствами требуется задействовать значительные вычислительные ресурсы микроконтроллера платы, в результате чего скорость и эффективность такой связи ниже, чем реализованной аппаратными средствами.

Плата Arduino Leonardo и многие другие 32-разрядные платы (например, Arduino Zero, Metro M0 компании Adafruit и RedBoard Turbo компании SparkFun) оснащены вторым аппаратным последовательным портом. Плата Teensy 3 компании PJRC (**http://www.pjrc.com/teensy**) вдобавок к основному имеет три дополнительных аппаратных последовательных порта, а плата Teensy 4.0 — четыре.

Плата Arduino Mega оснащена четырьмя аппаратными последовательными портами, т. е. может одновременно взаимодействовать с четырьмя разными последовательными устройствами. Один из этих портов подключен к встроенному адаптеру USB-ТТЛ, но для использования других последовательных портов для подключения по USB-каналу необходимо использовать внешний адаптер USB-ТТЛ.

В табл. 4.1 приводится список контактов цифрового ввода/вывода, используемых в разных платах Arduino, и их клонов для последовательных портов.

Таблица 4.1. *Контакты цифрового ввода/вывода некоторых плат Arduino, используемые для последовательных портов*

Плата	Serial RX/TX	Serial1 RX/TX	Serial2 RX/TX	Serial3 RX/TX
Arduino MKR 1010	Только с USB	13/14	Отсутствует	Отсутствует
Arduino Uno WiFi Rev2	Только с USB	0/1	Подключен к модулю Wi-Fi	Отсутствует
Arduino Nano Every	Только с USB	0/1	Отсутствует	Отсутствует
Arduino Nano 33 BLE Sense	Только с USB	0/1	Отсутствует	Отсутствует

Таблица 4.1 *(окончание)*

Плата	Serial RX/TX	Serial1 RX/TX	Serial2 RX/TX	Serial3 RX/TX
Arduino Uno Rev3	0/1 (также с USB)	Отсутствует	Отсутствует	Отсутствует
Adafruit Metro Express (M0)	Только с USB	0/1	Отсутствует	Отсутствует
Arduino Zero SparkFun RedBoard Turbo	Только с USB*	0/1	Отсутствует	Отсутствует
Adafruit Itsy Bitsy M4 Express	Только с USB	0/1	Отсутствует	Отсутствует
PJRC Teensy 3.2	Только с USB	0/1	9/10	7/8
PJRC Teensy 4.0	Только с USB	0/1	7/8	15/14
Arduino Due	0/1 (также с USB)	19/18	17/16	15/14
Arduino Mega 2560 Rev2	0/1 (также с USB)	19/18	17/16	15/14
Arduino Leonardo	Только с USB	0/1	Отсутствует	Отсутствует

* Используйте USB-Serial вместо последовательного порта.

Некоторые платы Teensy оснащены более чем тремя аппаратными портами, а другие позволяют изменять контакты, используемые для последовательной связи. Более подробная информация по платам Teensy предоставляется на веб-сайте компании-изготовителя PJRC (**https://oreil.ly/csezT**).

Особенности поведения аппаратных последовательных портов

Возможности последовательной связи разных плат отличаются не только количеством их аппаратных последовательных портов, но также некоторыми фундаментальными особенностями их работы. Большинство плат, оснащенных микроконтроллером AVR ATmega, включая плату Uno, оригинальную плату Nano и плату Mega, имеют дополнительный микроконтроллер, отвечающий за преобразование сигналов последовательного порта платы в сигналы USB для подключения к аппаратному последовательному порту компьютера через интерфейс USB. При открытии подключения к последовательному порту таких плат (например, запустив монитор порта среды Arduino IDE или обращаясь к последовательному порту платы через интерфейс USB из исполняющейся на компьютере программы) происходит автоматический сброс основного микроконтроллера платы, в результате чего загруженный в нее скетч начинает исполняться с начала.

В некоторых 8-разрядных платах (платах Leonardo и совместимых платах) и в большинстве 32-разрядных плат преобразование между интерфейсом USB и интерфейсом последовательного порта осуществляется основным (и единственным) микроконтроллером, отвечающим за исполнение загруженных в плату скетчей. Вследствие особенностей их конструкции открытие последовательного порта, пре-

образуемого в интерфейс USB[1], не вызывает сброса этих плат. В результате загруженный скетч начнет отправлять данные на порт USB-Serial намного быстрее, прежде чем пользователь подключится к последовательному порту на компьютере. Это означает, что если функция `setup()` скетча содержит функции последовательного вывода (`Serial.print()` или `Serial.println()`), отправляемые ими данные не будут отображаться в мониторе порта, поскольку вы не сможете запустить эту программу достаточно быстро (эту проблему можно решить, добавив задержку в функцию `setup()`, но существует более практичный способ, который мы рассмотрим далее).

Кроме этого, платы Leonardo и совместимые платы обладают еще одной особенностью, усложняющей работу с портом USB-Serial. В частности, при подаче питания на такую плату она переходит в специальный режим загрузки скетча через USB-подключение, длящийся несколько секунд. Этот режим обозначается миганием встроенного светодиода. В течение этого периода открыть последовательный порт будет невозможно.

Чтобы разрешить открытие последовательного порта для плат, в которых при открытии последовательного порта автоматический сброс не выполняется, в функцию `setup()` скетча можно добавить (сразу же после вызова функции `Serial.begin()`) приведенный далее код. Этот код не позволяет исполнение следующего за ним кода до тех пор, пока не будет открыт последовательный порт, что позволит отобразить в окне монитора порта информацию, посылаемую через порт USB-Serial в функции `setup()`:

```
while(!Serial)
{
   ; // Ожидаем открытие последовательного порта
}
```

Этот код можно свести к форме: `while(!Serial);`, но эта форма может сбивать с толку начинающих программистов.

Поскольку команда `while(!Serial);` приостанавливает дальнейшее исполнение скетча, пока не будет открыт последовательный порт, этот подход не следует использовать в автономных решениях на плате Arduino, т. е. исполняющихся без подключения платы к компьютеру с питанием платы от сетевого адаптера или батареи. В табл. 4.2 приводятся особенности работы порта USB-Serial для разных плат.

Таблица 4.2. *Особенности работы порта USB-Serial для разных плат*

Плата	Команда `while(!Serial);` нужна?	Плата сбрасывается при открытии последовательного порта?
Arduino MKR 1010	Да	Нет
Arduino Uno WiFi Rev2	Нет	Да

[1] Для краткости аппаратный последовательный порт, сигналы которого преобразовываются в сигналы USB, в дальнейшем мы будем называть USB-Serial.

Таблица 4.2 (окончание)

Плата	Команда `while(!Serial);` нужна?	Плата сбрасывается при открытии последовательного порта?
Arduino Nano Every	Нет. Требует использования команды `delay(800);` после вызова функции `Serial.begin()`, и монитор порта необходимо открыть, прежде чем загружать скетч в плату, чтобы все отправляемые платой данные отобразились в окне монитора порта	Нет
Arduino Nano 33 BLE Sense	Да	Нет
Arduino Uno Rev3	Нет	Да
Adafruit Metro Express (M0)	Да	Нет
Adafruit Itsy Bitsy M4 Express	Да	Нет
PJRC Teensy 3.2	Да	Нет
PJRC Teensy 4.0	Да	Нет
Arduino Mega 2560 Rev3	Нет	Да
Arduino Leonardo	Да	Нет

Эмуляция аппаратного последовательного порта посредством цифровых контактов ввода/вывода

Работа с аппаратными последовательными портами обычно осуществляется с помощью встроенной в среду Arduino IDE программной библиотеки Serial. Эта библиотека упрощает работу с последовательным портом, скрывая от разработчика большинство технических деталей аппаратного порта.

Иногда может возникнуть необходимость в большем количестве последовательных каналов связи, чем количество доступных аппаратных последовательных портов. Эту проблему можно решить с помощью дополнительной библиотеки SoftwareSerial, позволяющей программно эмулировать аппаратный последовательный порт. Использование библиотеки SoftwareSerial для одновременного взаимодействия с несколькими устройствами рассматривается в *разд. 4.11* и *4.12*.

Протоколы обмена сообщениями

Библиотеки для аппаратных и программных последовательных портов осуществляют отправку и прием информации, часто состоящей из групп переменных, которые необходимо отправлять вместе. Для правильной интерпретации такой информации принимающая сторона должна знать, где начинается и заканчивается каждое сообщение. Вразумительная последовательная связь и вообще любая связь между устройствами возможна лишь при полном согласовании передающей и принимаю-

щей сторонами организации информации в сообщении. Формальная организация информации в сообщении и диапазон допустимых ответов на запросы называются *протоколом обмена данными*. Протокол обмена данными можно установить поверх любой базовой системы передачи данных — например, последовательного интерфейса, но точно такие же принципы применимы и к любым другим средствам передачи данных, например по сети Ethernet или Wi-Fi.

Сообщения могут содержать один или несколько специальных символов, идентифицирующих начало сообщения. Эти символы называются *заголовком сообщения* (header). Конец сообщения также обозначается одним или несколькими символами, которые называются *окончанием сообщения* (footer).

В примерах этой главы рассматриваются варианты сообщений, в которых значения тела сообщения могут отправляться или в текстовом, или в двоичном формате:

- передача сообщений в *текстовом формате* осуществляется посредством отправки команд и числовых значений в виде букв и слов, которые могут понимать люди. Числа отправляются в виде строк цифровых символов, представляющих значение. Например, значение 1234 передается в виде отдельных цифровых символов: 1, 2, 3 и 4;
- сообщения в *двоичном формате* состоят из байтов, используемых для представления значений компьютерами. Передача данных в двоичном формате обычно более эффективна (требуется меньшее количество байтов), но людям сложно воспринимать данные в этом формате, что усложняет диагностирование возможных проблем с обменом. Например, число 1234 представляется в Arduino двумя байтами со значениями: 4 и 210 (4 * 256 + 210 = 1234). В мониторе порта эти символы не будут нести никакой значащей для людей информации, поскольку код ASCII 4 представляет управляющий символ, а код 210 содержится в расширенном наборе символов ASCII и, скорее всего, отобразится в виде какой-нибудь буквы с ударением или какого-либо специального символа, в зависимости от конфигурации вашего компьютера. Если подключаемое устройство может отправлять или получать только двоичные данные, у нас не будет иного выбора, кроме как работать с такими данными. Но если оно способно также отправлять данные в текстовом формате, то предпочтительнее выбрать такой формат, поскольку отправку и отладку текстовых сообщений легче организовать.

Задачу передачи данных можно решить многими программными способами, и в некоторых примерах этой главы приводятся два или три способа, дающих одинаковый результат. Выбор формата передачи данных (т. е. текстового или двоичного) может зависеть от текущих требований к простоте и эффективности передачи. При наличии такого выбора следует использовать тот формат, который легче поддается пониманию и реализации. Он обычно и будет рассматриваться в примерах первым. Альтернативный выбор может быть несколько более эффективным или более подходящим для конкретного протокола, используемого устройством, с которым нужно организовать обмен данными, но правильным способом будет тот, который легче всего поддается реализации.

Среда разработки Processing

В некоторых примерах этой главы для реализации обмена данными по последовательному каналу связи между компьютером и подключенной к нему платой Arduino используется язык программирования Processing.

Язык программирования Processing представляет собой свободно распространяемое средство с открытым исходным кодом, использующее среду разработки, подобную среде Arduino IDE, но разработанные на нем скетчи исполняются не на плате Arduino, а на компьютере. Дополнительную информацию по этому языку программирования можно найти на его вебсайте (**http://processing.org**). Там же можно и загрузить установочный пакет.

Язык программирования Processing основан на языке Java, но образцы кода на Processing в этой книге можно легко преобразовать для использования в других средах, поддерживающих последовательную связь. Среда разработки Processing содержит скетчи примеров, иллюстрирующих реализацию последовательной связи между платой Arduino и компьютером. Одним из скетчей Processing для связи с платой Arduino по последовательному каналу связи является скетч SimpleRead. Этот скетч получает по последовательному каналу связи данные от платы Arduino о состоянии подключенной к плате кнопки, и при кратковременном нажатии этой кнопки меняет цвет прямоугольника, отображаемого скетчем на экране компьютера. Открыть этот скетч и просмотреть его содержимое можно, выполнив команду меню Processing **Файл | Примеры**, в открывшемся окне **Примеры Java** развернуть узел **Библиотеки**, в нем развернуть узел **Serial** и выполнить двойной щелчок на примере **SimpleRead**.

Замечания по реализации последовательной связи в скетчах Arduino

Далее приводится список основных моментов, которые следует иметь в виду при работе с последовательной передачей данных в скетчах Arduino:

- функция `Serial.flush()` ожидает, пока не будут отправлены все исходящие данные, а не выбрасывает полученные данные (как она делала в более ранних версиях языка Arduino). Удалить все данные из буфера приема можно следующей строкой кода:

```
while(Serial.read() >= 0) ; // Очищаем буфер приема
```

- функции `Serial.write()` и `Serial.print()` не блокируют исполнение следующего за ними кода. В более ранних версиях языка Arduino возврат из этих функций выполнялся только после того, как они отправили все переданные им символы. Теперь же функции `Serial.write()`, `Serial.print()` и `Serial.println()` отправляют переданные им символы в фоновом режиме, используя обработчик прерывания, позволяя немедленное исполнение следующего за ними кода. Обычно это положительное свойство, поскольку улучшает время отклика скетча, но иногда требуется подождать, пока не будут отправлены все символы. В таком случае можно воспользоваться функцией `Serial.flush()`, вызвав ее сразу же после вызова любой из этих функций;
- функции последовательного вывода возвращают значение количества отправленных на вывод символов. Это свойство может быть полезным, когда необходимо выровнять отправленный текст или когда приложение отправляет данные, содержащие общее количество отправленных символов;

- язык Arduino содержит встроенное средство парсинга потоков, включая потоки последовательных данных, позволяющее с легкостью извлекать числа и определять текст. Дополнительная информация по использованию этой возможности с потоками последовательного обмена данными содержится в *разд. «Обсуждение работы решения и возможных проблем» разд. 4.5*;
- входящая в состав среды Arduino IDE библиотека SoftwareSerial может быть очень полезной, как показано в *разд. 4.11* и *4.12*;
- функция `Serial.peek()` позволят «подсмотреть» следующий символ в буфере приема. Но, в отличие от функции `Serial.read()`, применение этой функции не удаляет этот символ из буфера.

4.1. Передача данных скетчем Arduino на компьютер

ЗАДАЧА

Отправить текст и данные с платы Arduino на настольный компьютер или другое устройство (например, одноплатный компьютер Raspberry Pi), отображая их в окне монитора порта среды Arduino IDE или какой-либо другой программы терминала.

РЕШЕНИЕ

Следующий скетч (листинг 4.1) отправляет текстовые и числовые данные с платы Arduino и отображает их в окне монитора порта.

Листинг 4.1. Скетч для отправки данных по последовательному каналу связи

```
/*
* Скетч SerialOutput
* Выводит текстовые и числовые данные в окно монитора порта
*/

void setup()
{
   Serial.begin(9600); // Задаем скорость обмена величиной 9600 бод
}
int number = 0;

void loop()
{
   Serial.print("The number is ");
   Serial.println(number); // Выводим число на экран
   delay(500);             // Пауза в полсекунды перед выводом
                           // следующего числа
   number++;               // Переходим к выводу следующего числа
}
```

Подключите плату Arduino к компьютеру, как вы это делали в *главе 1*, и загрузите в нее этот скетч. Запустите монитор порта, щелкнув на его значке справа на панели инструментов среды Arduino IDE. Спустя короткое время в окне монитора порта должны начать выводиться отправляемые скетчем данные, как показано далее и на рис. 4.2.

```
The number is 0
The number is 1
The number is 2
```

Рис. 4.2. Вывод данных в окно монитора порта среды Arduino IDE

Обсуждение работы решения и возможных проблем

Для решения поставленной задачи передачи данных на компьютер по последовательному каналу связи мы сначала инициализируем библиотеку Serial, а затем посредством функции `Serial.print()` этой библиотеки передаем на компьютер требуемый текст и значения. Отправляемые данные отображаются на компьютере в окне монитора порта, как показано на рис. 4.2.

Среда Arduino IDE также содержит инструмент **Плоттер по последовательному порту**, позволяющий создавать график на основе принимаемых по последовательному порту числовых данных. Чтобы запустить этот инструмент, закройте окно монитора порта и выполните команду меню **Инструменты | Плоттер по последовательному порту**. Откроется окно плоттера, в котором принимаемые по последовательному каналу числовые данные будут отображаться в виде графика. Плоттер может отделять числа от текста, а также определять множественные числа, разделенные буквенными символами, и отдельно выводить их в графике, используя разные цвета. На рис. 4.3 показано окно плоттера с отображаемыми в нем данными, получаемыми от платы Arduino.

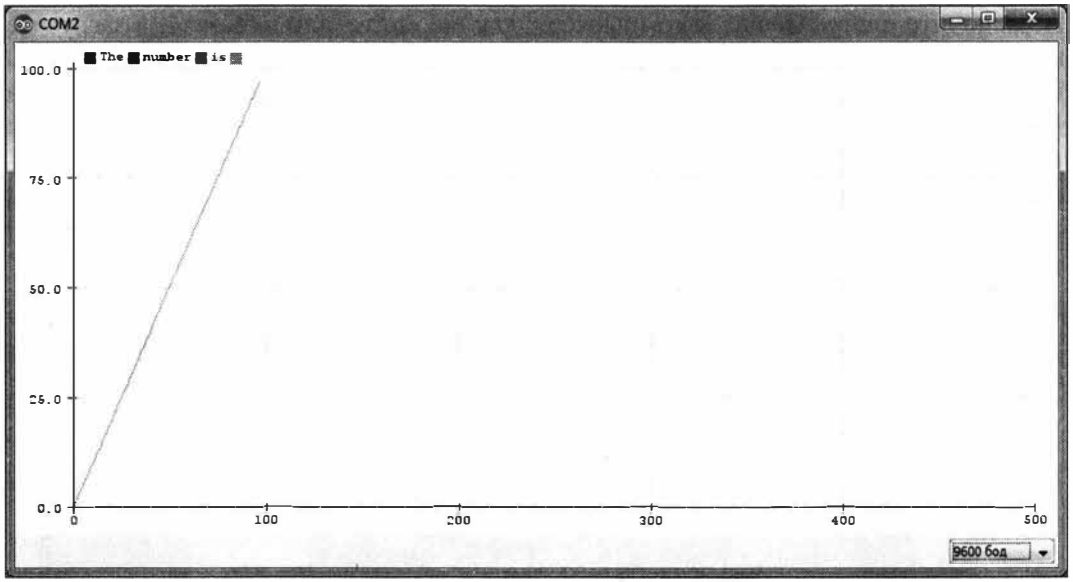


Рис. 4.3. Отображение данных в окне плоттера по последовательному порту

Чтобы скетч мог отправлять данные по последовательному каналу связи, необходимо вызвать функцию `Serial.begin()`. Ей передается только один параметр: требуемая скорость обмена данными. Эта скорость должна быть одинаковой как на передающей стороне, так и на принимающей, иначе в окне монитора порта будут выводиться бессмысленные символы (или вообще ничего не будет выводиться). В только что рассмотренном примере и в большинстве других примеров этой книги используется скорость в 9600 бод (термин *бод* означает бит в секунду). Поскольку один символ состоит из восьми битов, то скорость в 9600 бод означает приблизительно 1000 символов в секунду. Можно задавать более низкую или высокую скорость передачи (в диапазоне от 300 до 115 200 бод — в зависимости от возможностей используемой платы), но обе стороны должны быть настроены на одинаковую скорость. В мониторе порта скорость передачи устанавливается во втором справа выпадающем списке в правой нижней части окна монитора порта (см. рис. 4.2). Если выводимый в окне монитора порта текст выглядит наподобие этого:

```
`3??f<IxIOOOu`3??f<
```

то необходимо проверить, что заданная в мониторе порта скорость передачи совпадает со скоростью, заданной в вызове функции `Serial.begin()` в скетче.

Если в скетче и в мониторе порта задана одинаковая скорость передачи, но в мониторе порта все равно выводится нечитаемый текст, проверьте, что указана правильная плата (меню **Инструменты | Плата**). Некоторые платы имеют несколько версий с разной скоростью, поэтому, если указана неправильная версия, выберите правильную и повторно загрузите в нее скетч.

Собственно передача данных осуществляется посредством функции `Serial.print()`, в параметре которой указываются данные для передачи. Указанные в параметре

строки (текст в двойных кавычках) отображается в окне монитора порта как есть, но без кавычек. Например, следующий код:

```
Serial.print("The number is ");
```

выводит этот текст:

```
The number is
```

Формат вывода числовых значений зависит от типа переменной. Этот момент рассматривается более подробно в *разд. 4.2*. Например, для целочисленных переменных (тип `int`) выводятся их числовые значения. Так, если значение переменной `number` равно 1, то следующий код:

```
Serial.println(number);
```

выведет на экран это значение:

```
1
```

В скетче, приведенном в листинге 4.1, при первой итерации цикла выводится значение 0, и с каждой последующей итерацией это значение увеличивается на 1. В результате использования функции `Serial.println()`, а не `Serial.print()`, каждое последующее значение выводится в новой строке.

Следует иметь в виду две разные особенности поведения последовательного порта плат Arduino и совместимых плат. В частности, при открытии последовательного порта платы Uno большинство других плат AVR с микроконтроллером ATmega выполняют сброс. Это означает, что в мониторе порта или в плоттере данные всегда будут начинать отображаться с начального значения, то есть с 0. Но в платах Arduino Leonardo, а также в платах с процессорами ARM при открытии последовательного порта сброс платы не вызывается. Это означает, что скетч начнет счет немедленно после подачи питания на плату. В результате этого значение, выводимое в окно монитора порта или плоттера после его запуска, будет зависеть от момента запуска этих инструментов. В частности, чем позже после включения платы Arduino происходит запуск монитора порта, тем больше будет начальное значение, выводимое в нем. Более подробная информация на этот счет была приведена ранее, в *разд. «Особенности поведения аппаратных последовательных портов»*.

Этой информации должно быть достаточно, чтобы вы могли выводить в мониторе порта текст и целочисленные значения. Более подробная информация по форматированию выводимых данных приводится в *разд. 4.2*.

Кроме монитора порта среды Arduino IDE для обмена данными с платой Arduino можно использовать какую-либо программу терминала сторонних разработчиков. Кроме отображения данных в текстовом или двоичном формате (или и в том и в другом) многие такие программы предоставляют дополнительные возможности — например, отображение управляющих символов или запись получаемых данных в файл. Далее приводится список нескольких таких программ, рекомендуемых пользователями Arduino:

- CoolTerm (**https://oreil.ly/dk66x**) — удобная и бесплатная программа терминала для Windows, macOS и Linux;

- CuteCom (**https://oreil.ly/HnoJd**) — программа терминала с открытым исходным кодом для Linux;
- Bray Terminal (**https://oreil.ly/0Sm7j**) — бесплатная программа терминала для Windows, не требующая установки;
- GNU screen (**https://oreil.ly/5u8Di**) — программа с открытым исходным кодом для управления виртуальными экранами, поддерживающая последовательный обмен данными. Входит в состав дистрибутивов Linux и macOS;
- moserial (**https://oreil.ly/HCyPU**) — еще одна бесплатная программа терминала с отрытым исходным кодом для Linux;
- PuTTY (**https://oreil.ly/m6mpW**) — программа SSH с открытым исходным кодом, поддерживающая последовательный обмен данными для Windows и Linux;
- RealTerm (**https://oreil.ly/q9cAq**) — программа терминала с открытым исходным кодом для Windows;
- ZTerm (**https://oreil.ly/Ebb1B**) — условно бесплатная программа терминала для macOS.

В качестве выходного устройства для отображения данных, передаваемых по последовательному порту, можно использовать жидкокристаллический дисплей (ЖКД), однако функциональность такого дисплея будет очень ограниченной. При использовании ЖКД для вывода последовательно передаваемых данных посмотрите в документации на дисплей, как он обрабатывает возврат каретки, поскольку некоторые дисплеи не переходят автоматически на следующую строку при выводе данных, отправленных посредством функции `Serial.println()`. Кроме того, ЖКД подключается к плате Arduino через контакты TTL-Serial (цифровые контакты ввода/вывода 0 и 1), а не через USB-разъем. Для большинства плат AVR ATmega, таких как Uno, эти контакты соответствуют объекту `Serial`, поэтому с такими платами код из листинга 4.1 можно использовать без изменений. Но в платах Leonardo и некоторых платах, оснащенных процессором с ARM-архитектурой (например, платы с процессорами SAMD), контакты 0 и 1 соответствуют объекту `Serial1`, поэтому для них ключевое слово `Serial` в коде листинга 4.1 нужно заменить словом `Serial1`. В табл. 4.1 (*см. ранее*) приводится список конфигураций контактов объекта `Serial` для разных плат.

Дополнительная информация

Библиотека LiquidCrystal среды Arduino IDE для работы с ЖКД использует базовую функциональность вывода, подобную функциональности библиотеки Serial, поэтому многие из идей, рассматриваемых в этой главе, также применимы и к этой библиотеке. Работа с разного вида дисплеями, включая жидкокристаллические дисплеи, рассматривается более подробно в *главе 11*.

4.2. Отправка форматированного текста и числовых данных с платы Arduino

ЗАДАЧА

Отображать данные, отправляемые с платы Arduino по последовательному каналу связи, — такие как текст, десятичные числовые значения, шестнадцатеричные числовые значения или двоичные числовые значения.

РЕШЕНИЕ

Данные по последовательному порту можно передавать во многих разных форматах. В листинге 4.2 приводится скетч, демонстрирующий все опции форматирования данных, отправляемых функциями `Serial.print()` и `Serial.println()`.

Листинг 4.2. Скетч для передачи данных в разных форматах

```
/*
Скетч SerialFormatting
Передает данные в разных форматах на последовательный порт
*/

char chrValue = 65;   // Начальные передаваемые значения
byte byteValue = 65;
int intValue = 65;
float floatValue = 65.0;
char c1 = 4;
char c2 = 210;

void setup()
{
   while(!Serial); // Для плат Leonardo и с процессорами SAMD
                   // ожидаем открытия порта
   Serial.begin(9600);
}

void loop()
{
   Serial.print("chrValue: ");
   Serial.print(chrValue); Serial.print(" ");
   Serial.write(chrValue); Serial.print(" ");
   Serial.print(chrValue, DEC);
   Serial.println();
   Serial.print("byteValue: ");
   Serial.print(byteValue); Serial.print(" ");
   Serial.write(byteValue); Serial.print(" ");
   Serial.print(byteValue, DEC);
```

```
  Serial.println();
  Serial.print("intValue: ");
  Serial.print(intValue); Serial.print(" ");
  Serial.print(intValue, DEC); Serial.print(" ");
  Serial.print(intValue, HEX); Serial.print(" ");
  Serial.print(intValue, OCT); Serial.print(" ");
  Serial.print(intValue, BIN);
  Serial.println();
  Serial.print("floatValue: ");
  Serial.println(floatValue);
  Serial.println();
  delay(1000);   // Секундная пауза перед выводом следующего числа
  chrValue++;    // Переходим к выводу следующего числа
  byteValue++;
  intValue++;
  floatValue += 1;
}
```

В мониторе порта передаваемые значения отображаются следующим образом:

```
chrValue: A A 65
byteValue: 65 A 65
intValue: 65 65 41 101 1000001
floatValue: 65.00
chrValue: B B 66
byteValue: 66 B 66
intValue: 66 66 42 102 1000010
floatValue: 66.00
...
```

Обсуждение работы решения и возможных проблем

Передача текстовых строк по последовательному каналу связи не представляет никакой сложности. Достаточно указать в параметре функции `Serial.print();` необходимый текст (в двойных кавычках) — например, так: `Serial.print("Hello, world!");`, и она передаст этот текст по последовательному порту на другой конец канала. Если после передачи строки требуется выполнить переход на новую строку, вместо функции `Serial.print()` воспользуйтесь функцией `Serial.println()`.

С числовыми значениями дело обстоит сложнее. Способ отображения байтовых и целочисленных значений зависит от типа переменной и необязательного параметра форматирования. Язык Arduino позволяет очень либерально обращаться со значениями разных типов данных (дополнительная информация по типам данных приведена в *разд. 2.2*). Но эта гибкость может создавать путаницу, поскольку даже если числовые значения похожи друг на друга, компилятор считает их отдельными типами с разным поведением. Например, одинаковые значения типа `char`, `byte` и `int` необязательно одинаково будут отображены в окне монитора порта.

Далее приводится несколько конкретных примеров переменных разных типов с одинаковыми значениями.

```
char asciiValue = 'A';      // Код ASCII для символа 'A' имеет числовое значение 65
char chrValue = 65;         // 8-битовый символ со знаком и также символ ASCII 'A'
byte byteValue = 65;        // 8-битовый символ без знака и также символ ASCII 'A'
int intValue = 65;          // 16-битовое целое число со знаком и значением 65
float floatValue = 65.0;    // число с плавающий запятой (точкой) и значением 65
```

В табл. 4.3 показаны форматы вывода одинакового значения разных типов данных посредством функций Arduino `print()` и `write()`.

***Таблица 4.3.** Результаты вывода одного значения разных типов данных*

Тип данных	print (значение)	print (значение, DEC)	write (значение)	print (значение, HEX)	print (значение, OCT)	print (значение, BIN)
char	A	65	A	41	101	1000001
byte	65	65	A	41	101	1000001
int	65	65	A	41	101	1000001
long	Формат вывода значения типа long такой же, как и типа int					
float	65.00	Для значений типа float форматирование не поддерживается				
double	65.00	Для платы Uno тип double такой же, как и тип float				

Начиная с версии 1.0 Arduino, оператор `Serial.print(val,BYTE);` более не поддерживается.

Если в вашем коде для переменных типа `byte` ожидается такое же поведение, как и для типа `char` (т. е. вывод в виде символов ASCII), нужно использовать оператор `Serial.write(val);`.

В скетче из листинга 4.2 каждый оператор вывода занимает отдельную строку кода. Такой подход может сделать вывод сложных сообщений слишком громоздким. Например, для вывода таким способом следующего сообщения:

```
At 5 seconds: speed = 17, distance = 120
```

потребуется следующий код:

```
Serial.print("At ");
Serial.print(t);
Serial.print(" seconds: speed = ");
```

```
Serial.print(s);
Serial.print(", distance = ");
Serial.println(d);
```

Это слишком много строк кода для одной строки сообщения. Код можно сделать менее громоздким, поместив несколько операторов вывода в одну строку кода:

```
Serial.print("At "); Serial.print(t); Serial.print(" seconds, speed = ");
Serial.print(s); Serial.print(", distance = "); Serial.println(d);
```

Кроме того, для форматирования сообщений можно воспользоваться возможностью компилятора Arduino делать вставки в скомпилированный код. Для этого можно использовать некоторые усовершенствованные возможности языка C++ (синтаксис и шаблоны потоковых вставок), объявив в скетче потоковый шаблон. Самым простым способом для этого будет включение библиотеки Streaming, разработанной Микалом Хартом (Mikal Hart). Более подробная информация об этой библиотеке приводится на веб-сайте Микала (**https://oreil.ly/PJWJR**), а установить ее можно с помощью менеджера библиотек среды Arduino IDE, как описано в *разд. 16.2*.

При использовании библиотеки Streaming следующий код дает такой же результат, что и множественные операторы, показанные ранее:

```
Serial << "At " << t << " seconds, speed=" << s << ", distance=" << d << endl;
```

Опытные программисты могут задаться вопросом, почему язык Arduino не поддерживает оператор `printf`. Частично это объясняется использованием этим оператором динамической памяти и недостатком памяти RAM на 8-разрядных платах. Хотя более новые 32-разрядные платы обладают достаточным объемом памяти, команда разработчиков Arduino не хотела добавлять в возможности последовательного обмена языка Arduino такой краткий синтаксис, в котором очень легко допустить ошибки.

И тем не менее, хотя язык Arduino и не поддерживает оператор `printf`, в нем можно использовать родственный оператор `sprintf` для помещения отформатированного текста в буфер, который затем можно отправить на вывод с помощью оператора: `Serial.print/println`. Вот пример использования этого оператора:

```
char buf[100];
sprintf(buf, "At %d seconds, speed = %d, distance = %d", t, s, d);
Serial.println(buf);
```

Однако оператор `sprintf` может быть опасным, поскольку способен вызвать переполнение буфера, если размер записываемой в него строки превышает размер этого буфера. Куда в памяти будут записаны лишние символы, никому не ведомо, но почти наверняка они вызовут фатальный сбой или неправильное исполнение скетча. Такого нежелательного развития событий можно избежать, используя функцию `snprintf()`, в одном из параметров которой указывается максимальное количество записываемых в буфер символов (включая символ пробела, завершающий все строки). Это количество можно указать непосредственно числом элементов массива буфера (в нашем случае 100), но тогда следует не забыть изменить это число при изменении объема буфера. Лучше всего при этом воспользоваться оператором

sizeof(), чтобы определить объем буфера. Хотя во всех случаях, которые мы можем себе представить, значение типа char занимает один байт, чтобы получить размер массива, рекомендуется разделить размер массива на размер типа содержащихся в нем данных, как показано в следующем примере:

```
snprintf(buf, sizeof (buf) / sizeof (buf[0]),
"At %d seconds, speed = %d, distance = %d", t, s, d);
Serial.println(buf);
```

Хотя мы знаем, что для типа данных char значение sizeof (buf[0]) гарантированно будет 1, весьма полезно выработать такую привычку на случай использования других типов данных. Например, результатом исполнения следующего кода будет число 400:

```
long buf2[100];
Serial.println(sizeof (buf2));
```

А правильный результат можно получить, применив:

```
sizeof (buf2) / sizeof (buf2[0])
```

Но за удобство, предоставляемое использованием функций sprintf() и snprintf(), приходится расплачиваться памятью. Прежде всего, накладные расходы памяти возникают при создании буфера, объем которого в нашем случае составляет 100 байтов. Кроме того, дополнительные накладные расходы памяти создаются компиляцией этой функциональности в скетч. Для платы Arduino добавление такого кода повышает объем используемой памяти на 1648 байтов, что составляет 5 процентов памяти платы.

Дополнительная информация

Дополнительная информация по используемым в Arduino типам данных приводится в *главе 2*. На веб-сайте Arduino предлагаются справочные материалы по командам последовательного обмена данными (**https://oreil.ly/8MGsb**), а также по выводу в поток (**https://oreil.ly/JqHPa**).

4.3. Прием последовательно передаваемых данных платой Arduino

ЗАДАЧА

Необходимо принимать на плате Arduino сообщения, передаваемые по последовательному каналу связи с компьютера или другого устройства, — например, команды или данные.

РЕШЕНИЕ

Пример скетча для решения этой задачи приводится в листинге 4.3. Этот скетч принимает цифру (числовое значение от 0 до 9) и мигает светодиодом с частотой, пропорциональной значению принятой цифры.

Листинг 4.3. Скетч приема данных по последовательному каналу

```
/*
* Скетч SerialReceive
* Мигает светодиодом с частотой, пропорциональной величине принятой цифры
*/
int blinkDelay = 0; // Переменная для хранения задержки для мигания

void setup()
{
   Serial.begin(9600); // Задаем скорость обмена по последовательному
                       // порту 9600 бод
   pinMode(LED_BUILTIN, OUTPUT); // Задаем режим вывода для этого контакта
}

void loop()
{
   if (Serial.available()) // Проверяем наличие хотя бы одного принятого символа
   {
      char ch = (char) Serial.read();
      if( isDigit(ch) ) // Проверяем, что принят символ ASCII от 0 до 9
         {
            blinkDelay = (ch - '0'); // Преобразовываем значение
                                     // ASCII в числовое значение
            blinkDelay = blinkDelay * 100; // Величина задержки равна
                                           // 100 мс * принятую цифру
         }
   }

   blink();
}
// Мигаем светодиодом с длительностью включенного и выключенного
// состояний, заданной значением переменной blinkDelay

void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
   delay(blinkDelay); // Задержка определяется значением
                      // переменной blinkDelay
   digitalWrite(LED_BUILTIN, LOW);
   delay(blinkDelay);
}
```

Загрузите скетч в плату и посылайте на плату сообщения, используя монитор порта. Запустите монитор порта, введите какой-либо символ в текстовое поле вверху окна и нажмите кнопку **Отправить** (или клавишу <Enter>). Введенный символ

будет отправлен на плату Arduino. Если этот символ — цифра, то частота мигания светодиода должна измениться: чем больше цифра, тем медленнее мигание.

Обсуждение работы решения и возможных проблем

Функция `Serial.read()` возвращает целочисленное значение (тип `int`), поэтому его необходимо преобразовать в тип `char`, чтобы его можно было использовать в последующих операциях сравнения. Если вам не знаком способ представления символов кодами ASCII, то процесс преобразования принятых символов ASCII в числовые значения может быть для вас не совсем очевидным. Следующий код преобразует значение кода ASCII для цифрового символа, содержащееся в переменной `ch`, в соответствующее числовое значение:

```
blinkDelay = (ch - '0'); // Преобразовываем значение кода ASCII
                         // для цифры в соответствующее числовое значение
```

Цифровые символы от «0» до «9» представляются кодами ASCII от 48 до 57 (см. *приложение 7*). Чтобы преобразовать цифровой символ ASCII в соответствующее числовое значение, от кода ASCII для этого символа (от 48 до 57) нужно отнять код ASCII для символа «0» (равен 48). Например, если переменная `ch` содержит цифровой символ «1», его код ASCII будет 49, а его числовое значение равно 49 – 48 = 1. Выражение `49 - '0'` равнозначно 49 – 48 = 1, что соответствует числовому значению цифрового символа «1».

Иными словами, выражение `(ch - '0')` равнозначно выражению `(ch - 48)` и преобразовывает значение кода ASCII для цифрового символа, находящегося в переменной `ch`, в соответствующее числовое значение.

Числа, состоящие из более чем одной цифры, можно получать с помощью функций `parseInt()` и `parseFloat()`, которые упрощают извлечение числовых значений из потока последовательно передаваемых данных (эти функции также применимы с объектами `Ethernet` и другими объектами класса `Stream`. Дополнительная информация по использованию сетевых объектов для парсинга потоковых данных приводится во введении в *главу 15* (*разд. 15.0*)).

Функции `Serial.parseInt()` и `Serial.parseFloat()` считывают цифровые символы из последовательного потока данных и возвращают их соответствующие числовые значения (листинг 4.4). Нецифровые символы, предшествующие цифровым, игнорируются, а первый нецифровой символ (или символ точки в случае функции `parseFloat()`), следующий за цифровым, указывает на завершение этого числа. Если поток входящих данных не содержит никаких цифровых символов, функции возвращают значение 0, поэтому следует выполнять проверку на нулевые значения и обрабатывать их должным образом. Если же в настройках монитора порта указано завершение сообщения символом новой строки или возврата каретки (или обоими этим символами), то функции `parseInt()` и `parseFloat()` будут пытаться интерпретировать эти символы как цифровые и, потерпев неудачу, возвратят нуль. В результате переменной `blinkDelay` будет присвоено нулевое значение сразу же после присвоения ей желаемого значения и светодиод не будет мигать, какие бы значения мы ни отправляли в плату Arduino.

Листинг 4.4. Скетч приема данных по последовательному каналу с использованием функции `parseInt()`

```
/*
* Скетч SerialParsing
* Мигает светодиодом с частотой, пропорциональной величине принятой цифры
*/
int blinkDelay = 0;
void setup()
{
   Serial.begin(9600); // Задаем скорость обмена по последовательному
                       // порту 9600 бод
   pinMode(LED_BUILTIN, OUTPUT); // Задаем режим вывода для этого контакта
}

void loop()
{
   if ( Serial.available()) // Проверяем наличие хотя бы одного принятого символа
   {
      int i = Serial.parseInt();
      if (i != 0)
      {
         blinkDelay = i;
      }
   }
blink();
}
// Мигаем светодиодом с длительностью включенного и выключенного
// состояний, заданной значением переменной blinkDelay

void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
   delay(blinkDelay); // Задержка определяется значением переменной blinkDelay
   digitalWrite(LED_BUILTIN, LOW);
   delay(blinkDelay);
}
```

В *разд. 4.5* рассматривается еще один случай использования функции `parseInt()` для нахождения и извлечения чисел из последовательного потока данных.

Другой подход к преобразованию текстовых строк, представляющих числа, основан на использовании функций языка C `atoi()` (для переменных типа `int`) или `atol()` (для переменных типа `long`). Эти функции с малопонятными названиями (которые будут пояснены чуть позже) преобразовывают строки в целые числа типа `int` и `long`. Они предоставляют более расширенные возможности для манипулирования входящими данными, но за счет повышенной сложности кода. В частности, прежде

чем вызывать эти функции преобразования, необходимо принять всю строку и сохранить ее в массиве.

В листинге 4.5 приводится фрагмент кода, который использует функцию `atoi()` для извлечения из входящего последовательного потока данных числовых значений частоты мигания светодиода. Конец строки символов значения задержки указывается нецифровым символом (или заполнением буфера).

Листинг 4.5. Использование функции `atoi()` для извлечения значения задержки

```
const int maxChars = 5; // Переменная максимального количества символов
                        // значения задержки
char strValue[maxChars+1]; // Массив для хранения символов значения задержки.
                       // Он должен иметь достаточно места для хранения самих символов
                       // плюс завершающего нуля
int idx = 0;  // Указатель для массива, хранящего полученные символы
              // значения задержки

void loop()
{
   if( Serial.available())
   {
      char ch = (char) Serial.read();
      if( idx < maxChars && isDigit(ch)
      {
         strValue[idx++] = ch;    // Добавляем в строку символ ASCII
      }
      else
      {
         // Идем сюда, когда буфер заполнен или получен первый нецифровой символ
         strValue[idx] = 0; // Завершаем строку символом 0
         blinkDelay = atoi(strValue); // Используем функцию atoi()
         // для преобразования строки цифровых символов в соответствующее
         // числовое значение
         idx = 0;
      }
   }
blink();
}
```

Переменная `strValue` используется для построения строки символов значения задержки из цифровых символов, получаемых по последовательному каналу связи.

Дополнительная информация по символьным строкам приводится в *разд. 2.6*.

Функция `atoi()` (название расшифровывается как ASCII to integer, из ASCII в целое число) преобразовывает строку цифровых символов в соответствующее целочисленное значение (типа `int`). Функция `atol()` (ASCII to long) — делает то же самое, но только в значение типа `long`.

Язык Arduino также поддерживает функцию `serialEvent()` для обработки символов, получаемых по последовательному каналу связи. Код внутри этой функции вызывается при каждой итерации главной функции `loop()`. Код в листинге 4.6 выполняет ту же задачу, что и код в листинге 4.3, но для обработки входящих символов использует функцию `serialEvent()`.

Листинг 4.6. Обработка входящих символов с помощью функции `serialEvent()`

```
/*
* Скетч SerialEvent Receive
* Мигает светодиодом с частотой, пропорциональной величине принятой цифры
*/
int blinkDelay = 0; // Переменная для хранения задержки для мигания

void setup()
{
   Serial.begin(9600); // Задаем скорость обмена по последовательному порту 9600 бод
   pinMode(LED_BUILTIN, OUTPUT); // Задаем режим вывода для этого контакта
}

void loop()
{
   blink();
}

void serialEvent()
{
   while(Serial.available())
   {
      char ch = (char) Serial.read();
      Serial.write(ch);
      if( isDigit(ch) ) // Проверяем, что принят символ ASCII от 0 до 9
      {
         blinkDelay = (ch - '0'); // Преобразовываем значение кода ASCII
         // для цифры в соответствующее числовое значение
         blinkDelay = blinkDelay * 100; // Величина задержки равна
                                        // 100 мс * принятую цифру
      }
   }
}
// Мигаем светодиодом с длительностью включенного и выключенного
// состояний, заданной значением переменной blinkDelay
```

```
void blink()
{
   digitalWrite(LED_BUILTIN, HIGH);
   delay(blinkDelay); // Задержка определяется значением переменной blinkDelay
   digitalWrite(LED_BUILTIN, LOW);
   delay(blinkDelay);
}
```

Дополнительная информация

Дополнительную информацию по функциям `atoi()` и `atol()` можно найти в Интернете, выполнив поиск по этим названиям. Для начала можно ознакомиться со справкой по этим функциям в Википедии (**https://oreil.ly/8kQqp**).

4.4. Передача с платы Arduino нескольких текстовых строк в одном сообщении

ЗАДАЧА

Передать с платы Arduino сообщение, состоящее из нескольких отдельных элементов информации, — например, значений от нескольких датчиков. Эти значения нужно использовать в программе, исполняющейся на ПК или одноплатном компьютере Raspberry Pi.

РЕШЕНИЕ

Простой способ решения этой задачи — разделить отдельные элементы строки каким-либо символом разделения. Обычно для этого используется запятая. Пример скетча, составляющего строку из отдельных элементов и передающего ее, приводится в листинге 4.7.

Листинг 4.7. Скетч Arduino для передачи строки значений, разделенных запятыми

```
// Скетч CommaDelimitedOutput
void setup()
{
   Serial.begin(9600);
}

void loop()
{
   int value1 = 10;   // Произвольные значения для передачи
   int value2 = 100;
   int value3 = 1000;
   Serial.print('H'); // Уникальный заголовок, идентифицирующий начало сообщения
   Serial.print(",");
```

```
   Serial.print(value1,DEC);
   Serial.print(",");
   Serial.print(value2,DEC);
   Serial.print(",");
   Serial.print(value3,DEC);
   Serial.println(); // Возврат каретки и новая строка для завершения сообщения
   delay(100);
}
```

На компьютере передаваемые платой Arduino данные можно принимать с помощью следующего скетча Processing (листинг 4.8).

Листинг 4.8. Скетч Processing для приема строки значений, разделенных запятыми

```
// Скетч Processing для приема и обработки строки значений, разделенных запятыми.
// Ожидает входящую строку в формате:
// заголовок,значение1,значение2,значение3
import processing.serial.*;
Serial myPort; // Создаем экземпляр класса Serial
char HEADER = 'H'; // Символ, идентифицирующий начало сообщения
short LF = 10; // Символ новой строки ASCII
// ВНИМАНИЕ!
// Если необходимо, измените указанный в следующей строке номер порта на правильный
short portIndex = 0; // Задаем номер порта COM. Нумерация портов
                     // начинается со значения 0

void setup()
{
   size(200, 200);
   println( (Object[]) Serial.list());
   println(" Connecting to -> " + Serial.list()[portIndex]);
   myPort = new Serial(this, Serial.list()[portIndex], 9600);
}

void draw()
{
   if (myPort.available() > 0)
   {
      String message = myPort.readStringUntil(LF); // Считываем данные,
                              // получаемые по последовательному порту
      if (message != null)
      {
         message = message.trim(); // Удаляем символы пробелов
                                   // в начале и конце строки
         println(message);
         String [] data = message.split(","); // Разделяем полученное
                                              // сообщение по запятым
```

```
        if (data[0].charAt(0) == HEADER && data.length == 4) // Проверяем
                                                             // действительность
        {
           for (int i = 1; i < data.length; i++) // Пропускаем заголовок
                                                 // (начинаем с поля 1, а не 0)
           {
              println("Value " + i + " = " + data[i]); // Выводим
                                                       // на экран значения полей
           }
           println();
        }
     }
  }
}
```

Обсуждение работы решения и возможных проблем

Код скетча Arduino в листинге 4.7 отправляет на последовательный порт следующую текстовую строку (`\r` представляет возврат каретки, а `\n` — новую строку):

```
H,10,100,1000\r\n
```

В качестве разделительного символа необходимо использовать такой символ, который никогда не будет частью собственно данных. Например, если данные состоят только из числовых значений, в качестве разделительного символа можно использовать запятую. Неплохо также предоставить принимающей стороне возможность определить начало сообщения, чтобы удостовериться в наличии всех данных всех полей. Эта задача решается добавлением символа заголовка в начало сообщения. Как и разделитель, символ заголовка не может быть частью ни одного из полей данных и должен отличаться от символа разделителя. В приведенном примере начало сообщения указывается заглавной буквой `H` (английской). Таким образом, наше сообщение состоит из заголовка, трех строк ASCII числовых значений, разделенных запятыми, и символов возврата каретки и новой строки.

Символы возврата каретки и новой строки отправляются при каждом использовании функции `println()` и используются принимающей стороной, чтобы определить конец принимаемого сообщения. Поскольку код:

```
myPort.readStringUntil(LF)
```

в скетче Processing будет обрабатывать символ возврата каретки (`\r`), который предшествует символу новой строки, в скетче используется функция `trim()` для удаления ведущих и замыкающих непечатаемых символов, в число которых входят символы пробела, табуляции, возврата каретки и новой строки.

Скетч Processing считывает всю строку сообщения, а затем разбивает ее с помощью функции `split()` по запятым на отдельные поля, которые сохраняются в массиве.

В большинстве случаев на компьютерах macOS или Windows без физического последовательного порта следует использовать первый порт COM в списке портов. А на компьютерах с физическим последовательным портом следует использовать последний порт в списке. Скетч Processing содержит код, показывающий доступ-

ные последовательные порты и текущий выбранный порт, поэтому необходимо проверить, что подключение к плате Arduino осуществляется именно по этому порту. Если при первом исполнении скетча возникнет ошибка, проверьте список доступных портов в панели сообщений в нижней части окна Processing, чтобы определить, какой номер порта использовать для переменной `portIndex`[2].

Отображение значений датчиков с помощью кода Processing поможет визуализировать эти данные, что может сэкономить вам много часов времени на отладку. В то время как формат отображения данных CSV является полезным и широко распространенным, формат JSON (JavaScript Object Notation, система обозначений объектов JavaScript) более выразительный, а также более удобный для восприятия людьми. Формат обмена данными JSON широко используется для сетевого обмена сообщениями. Скетч в листинге 4.9 считывает выходной сигнал акселерометра платы Arduino WiFi Rev 2 или Arduino Nano 33 BLE Sense (раскомментируйте строку кода, соответствующую используемой вами одной из этих плат) и отправляет полученные значения в формате JSON на компьютер по последовательному каналу связи.

Листинг 4.9. Чтение акселерометра и передача полученных данных в формате JSON

```
/*
 Скетч AccelerometerToJSON.
 Отправляет полученные с акселерометра данные в формате JSON
 через последовательный порт на компьютер
*/
#include <Arduino_LSM6DS3.h>    // Плата Arduino WiFi R2
//#include <Arduino_LSM9DS1.h>  // Плата Arduino BLE Sense

void setup()
{
   Serial.begin(9600);
   while (!Serial);
   if (!IMU.begin())
   {
      while (1)
      {
         Serial.println("Error: Failed to initialize IMU");
         // Ошибка: Не удалось инициализировать блок инерциальных измерений
         delay(3000);
      }
   }
}
```

[2] При этом следует иметь в виду, что нумерация последовательных портов начинается со значения 0. Поэтому значение переменной `portIndex` для порта, например, COM2 будет 1, а не 2.

```
void loop()
{
   float x, y, z;
   if (IMU.accelerationAvailable())
   {
      IMU.readAcceleration(x, y, z);
      Serial.print("{");
      Serial.print("'x': "); Serial.print(x); Serial.print(", ");
      Serial.print("'y': "); Serial.print(y); Serial.print(", ");
      Serial.print("'z': "); Serial.print(z); Serial.print(", ");
      Serial.println("}");
      delay(200);
   }
}
```

На компьютере данные, посылаемые скетчем Arduino в листинге 4.9, принимаются и обрабатываются скетчем Processing, код которого приводится в листинге 4.10. Этот скетч отображает в режиме реального времени до 12 значений с плавающей запятой, получаемых от платы Arduino, в диапазоне от –5 до +5.

Листинг 4.10. Прием и обработка данных акселерометра, отправляемых платой Arduino

```
/*
* Скетч ShowSensorData.
*
* Отображает в полосовом индикаторе данные в формате JSON в диапазоне от -127 до 127
* Ожидает данные в формате "{'метка1': значение, 'метка2': значение,}\n"
* Например:
* {'x': 1.0, 'y': -1.0, 'z': 2.1,}
*/
import processing.serial.*;
import java.util.Set;
Serial myPort;          // Создаем экземпляр объекта класса Serial
PFont fontA;            // Шрифт для отображения текста
int fontSize = 12;
short LF = 10;          // Символ новой строки ASCII
int rectMargin = 40;
int windowWidth = 600;
int maxLabelCount = 12; // Используйте большее значение, если требуется больше меток
int windowHeight = rectMargin + (maxLabelCount + 1) * (fontSize *2);
int rectWidth = windowWidth - rectMargin*2;
int rectHeight = windowHeight - rectMargin;
int rectCenter = rectMargin + rectWidth / 2;
int origin = rectCenter;
int minValue = -5;
int maxValue = 5;
float scale = float(rectWidth) / (maxValue - minValue);
```

```
// ВНИМАНИЕ!
// Если необходимо, измените указанный в следующей строке номер порта на правильный
short portIndex = 0; // Задаем номер порта COM. Нумерация портов
                     // начинается со значения 0

void settings()
{
   size(windowWidth, windowHeight);
}

void setup()
{
   println( (Object[]) Serial.list());
   println(" Connecting to -> " + Serial.list()[portIndex]);
   myPort = new Serial(this, Serial.list()[portIndex], 9600);
   fontA = createFont("Arial.normal", fontSize);
   textFont(fontA);
}

void draw()
{
   if (myPort.available () > 0)
   {
      String message = myPort.readStringUntil(LF);
      if (message != null)
      {
         // Загружаем из сообщения данные в формате JSON
         JSONObject json = new JSONObject();
         try
         {
            json = parseJSONObject(message);
         }
         catch(Exception e)
         {
            println("Could not parse [" + message + "]");
           //Не удалось выполнить парсинг
         }
         // Копируем метки и значения в отдельные массивы
         ArrayList<String> labels = new ArrayList<String>();
         ArrayList<Float> values = new ArrayList<Float>();
         for (String key : (Set<String>) json.keys())
         {
            labels.add(key);
            values.add(json.getFloat(key));
         }
         // Прорисовываем координатную сетку и наносим на нее значения
         background(255);
         drawGrid(labels);
```

```
            fill(204);
            for (int i = 0; i < values.size(); i++)
            {
               drawBar(i, values.get(i));
            }
         }
      }
}

// Прорисовываем полосу, представляющую текущее значение датчика
void drawBar(int yIndex, float value)
{
   rect(origin, yPos(yIndex)-fontSize, value * scale, fontSize);
}

void drawGrid(ArrayList<String> sensorLabels)
{
   fill(0);
   // Прорисовываем метку минимального значения и вертикальную линию для него
   text(minValue, xPos(minValue), rectMargin-fontSize);
   line(xPos(minValue), rectMargin, xPos(minValue), rectHeight + fontSize);
   // Прорисовываем метку среднего значения и вертикальную линию для него
   text((minValue+maxValue)/2, rectCenter, rectMargin-fontSize);
   line(rectCenter, rectMargin, rectCenter, rectHeight + fontSize);
   // Прорисовываем метку максимального значения и вертикальную линию для него
   text(maxValue, xPos(maxValue), rectMargin-fontSize);
   line(
   xPos(maxValue), rectMargin, xPos(maxValue), rectHeight + fontSize);
   // Отображаем метку датчика каждой оси
   for (int i=0; i < sensorLabels.size(); i++)
   {
      text(sensorLabels.get(i), fontSize, yPos(i));
      text(sensorLabels.get(i), xPos(maxValue) + fontSize, yPos(i));
   }
}

// Вычисляем позицию по оси Y с учетом полей и размеров шрифтов
int yPos(int index)
{
   return rectMargin + fontSize + (index * fontSize*2);
}

// Вычисляем позицию по оси X с учетом полей и размеров шрифтов
int xPos(int value)
{
   return origin + int(scale * value);
}
```

На рис. 4.4 показан пример отображения полученных значений акселерометра для каждой оси (x, y, z) в виде полос, отходящих от средней линии при перемещении платы Arduino в пространстве.

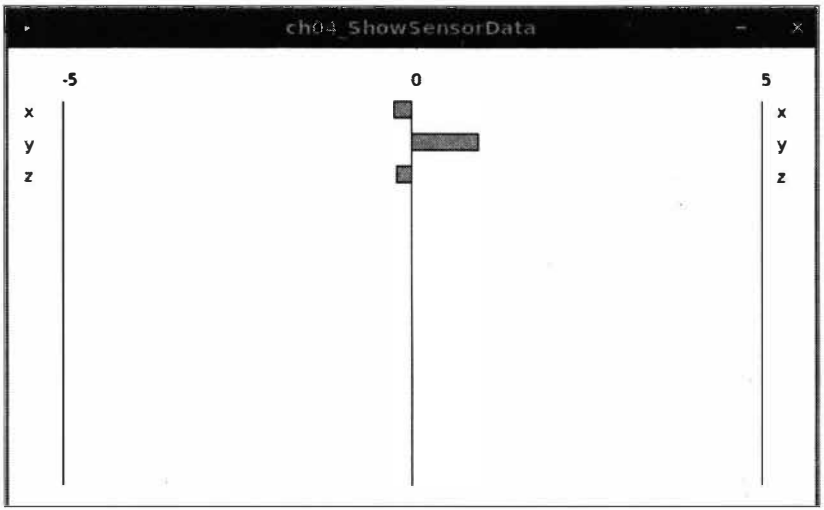


Рис. 4.4. Созданное скетчем Processing окно, отображающее полученные значения датчика

При необходимости значения и начало координат графика можно с легкостью изменить. Например, полосы графика можно отображать на левой оси для диапазонов значений от 0 до 1024 посредством следующего кода:

```
int origin = rectMargin; // Переменная rectMargin представляет
                         // левый край области графика
int minValue = 0;
int maxValue = 1024;
```

Если у вас нет акселерометра, значения для входных аналоговых значений можно генерировать с помощью скетча, код которого приводится в листинге 4.11. Отсутствующие датчики можно эмулировать, просто проводя пальцами внизу платы вдоль аналоговых контактов ввода/вывода, создавая уровни, которые можно просматривать с помощью скетча Processing. В этом скетче значения показаний занимают диапазон от 0 до 1023, поэтому для него в скетче Processing нужно изменить начало координат и минимальное и максимальное значения (см. листинг 4.10).

Листинг 4.11. Эмуляция показаний датчиков

```
/*
Скетч AnalogToJSON. Отправляет аналоговые данные в формате JSON
через последовательный порт на компьютер
*/

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

```
   while (!Serial);
}

void loop()
{
   Serial.print("{");
   Serial.print("'A0': "); Serial.print(analogRead(A0)); Serial.print(", ");
   Serial.print("'A1': "); Serial.print(analogRead(A1)); Serial.print(", ");
   Serial.print("'A2': "); Serial.print(analogRead(A2)); Serial.print(", ");
   Serial.print("'A3': "); Serial.print(analogRead(A3)); Serial.print(", ");
   Serial.print("'A4': "); Serial.print(analogRead(A4)); Serial.print(", ");
   Serial.print("'A5': "); Serial.print(analogRead(A5)); Serial.print(", ");
   Serial.println("}");
}
```

Дополнительная информация

Подробная информация по установке и использованию среды разработки Processing приводится на веб-сайте этой платформы (**http://processing.org**).

Кроме того, по среде разработки Processing существует обширная литература. Мы, например, можем порекомендовать вам следующие книги:

- Getting Started with Processing (Processing для начинающих), второе издание, авторы Casey Reas и Ben Fry, издательство Make.
- Processing: A Programming Handbook for Visual Designers and Artists (Язык Processing. Справочник по программированию для визуальных разработчиков и художников), авторы Casey Reas и Ben Fry, издательство MIT Press.
- Visualizing Data (Визуализация данных), автор Ben Fry, издательство O'Reilly.
- Processing: Creative Coding and Computational Art (Язык Processing. Творческое программирование и искусство вычислений), автор Ira Greenberg, издательство Apress.
- Making Things Talk, автор Tom Igoe, издательство Make Community. В книге рассматриваются платформа Arduino и язык Processing и приводится много примеров кода для обмена данными[3].

4.5. Прием платой Arduino нескольких текстовых полей в одном сообщении

ЗАДАЧА

Принять на плате Arduino сообщение, состоящее из нескольких отдельных элементов информации. Сообщение может содержать идентификатор определенного уст-

[3] Доступен русский перевод этой книги: Иго Том. Arduino, датчики и сети для связи устройств, 2-е изд., Издательство «БХВ-Петербург»: **https://bhv.ru/product/arduino-datchiki-i-seti-dlya-svyazi-ustrojstv-2-e-izd/**.

ройства (например, электродвигатель или какой-либо иной привод) и значение, которое следует присвоить этому устройству (например, скорость вращения).

РЕШЕНИЕ

Язык Arduino не имеет отдельной функции `split()`, доступной в языке Processing (см. *разд. 4.4*), но подобную функциональность можно реализовать с помощью скетча, код которого показан в листинге 4.12. Этот скетч принимает сообщение, содержащее заголовок в виде символа английской буквы `H`, за которым следуют три числовых поля, разделенных запятыми. Сообщение завершается символом новой строки.

Листинг 4.12. Разбиение принятого сообщения на составляющие поля

```
/*
* Скетч SerialReceiveMultipleFields
* Этот код ожидает сообщение в формате: H,12,345,678
* Код ожидает символ новой строки в качестве указателя конца данных
* В настройках монитора порта задайте параметр NL (Новая строка)
*/
const int NUMBER_OF_FIELDS = 3; // Количество ожидаемых полей, разделенных запятыми
int values[NUMBER_OF_FIELDS];   // Массив для хранения значений этих полей

void setup()
{
   Serial.begin(9600); // Задаем скорость обмена по последовательному порту 9600 бод
}

void loop()
{
   if ( Serial.available())
   {
      if (Serial.read() == 'H')
      {
         // Read the values
         for (int i = 0; i < NUMBER_OF_FIELDS; i++)
         {
            values[i] = Serial.parseInt();
         }
         // Отображаем значения в формате разделения запятыми
         Serial.print(values[0]); // Первое значение
         // Остальные значения отображаем с ведущей запятой
         for (int i = 1; i < NUMBER_OF_FIELDS; i++)
         {
            Serial.print(","); Serial.print(values[i]);
         }
```

```
        Serial.println();
      }
    }
}
```

Обсуждение работы решения и возможных проблем

В скетче используется функция `parseInt()`, позволяющая с легкостью извлекать числовые значения из последовательных потоков или потоков интернет-вещания. Рассматриваемый скетч — лишь один из примеров использования этой возможности. В *главе 15* приводятся дополнительные примеры парсинга потоковых данных. Проверить работу скетча можно, открыв монитор порта и отправив разделенное запятыми сообщение — например: `H1,2,3`. Функция `parseInt()` игнорирует все символы, кроме символа минус (-) и цифровых символов, поэтому в качестве разделителя не обязательно использовать запятую, вполне можно взять и какой-либо другой символ — например: `H1/2/3`. Скетч сохраняет извлеченные значения в массиве, а затем отправляет их, разделив запятыми, обратно на монитор порта.

Способ, которым скетч отображает список разделенных запятыми значений, поначалу может показаться необычным. Сначала отображается первое полученное число (например, 1), а затем выводятся остальные числа, каждое с ведущей запятой (например: ,2, а затем ,3). Можно было бы сэкономить несколько строк кода и выводить числа с запятой после каждого числа, но тогда вместо 1,2,3 получится 1,2,3,. Многие другие языки программирования, включая язык Processing, содержат функцию `join()`, которая составляет из элементов массива строку разделенных запятыми значений, но язык Arduino такой функции не имеет.

Функции парсинга потоковых данных ожидают следующий символ в течение определенного времени — по умолчанию время тайм-аута составляет одну секунду. Если в течение этого времени функция `parseInt()` не получит никаких символов, она возвращает значение 0. Время тайм-аута можно изменить, вызвав функцию:

```
Stream.setTimeout(время_тайм-аута)
```

и указав в параметре время тайм-аута. Время тайм-аута задается длинным целым числом в миллисекундах, т. е. диапазон времени тайм-аута составляет от 0 до 2147483647 мс. Следующий код задает максимальное время тайм-аута:

```
Stream.setTimeout(2147483647); // задаем время тайм-аута чуть меньше чем 25 дней
```

Далее приводится краткое описание функций парсинга потоковых данных, поддерживаемых языком Arduino (не все эти функции были использованы в предыдущем примере).

- `bool find(char *target);`

  Считывает данные из потока, пока не найдет указанное значение, возвращая булево значение `true` (истина). Если строка не содержит указанных данных, возвращается булево значение `false` (ложь). Обратите внимание: при парсинге потоковых данных по строке осуществляется только один проход. Возвращение обратно к строке, чтобы найти в ней какое-либо другое значение, не поддерживается (см. функцию `findUntil()`).

- `bool findUntil(char *target, char *terminate);`

  Эта функция похожа на функцию `find()`, но поиск завершается при обнаружении строки, указанной в параметре `terminate`. При этом булево значение `true` возвращается только в случае обнаружения целевой строки. Эта возможность позволяет прекратить поиск по обнаружению ключевого слова или разделителя. Например, следующий код:

  ```
  finder.findUntil("целевая_строка", "\n");
  ```

  будет пытаться найти значение параметра `"целевая_строка"`, но прекратит поиск при обнаружении символа завершения поиска, которым в рассматриваемом случае является символ новой строки `\n`.

- `long parseInt();`

  Возвращает первое действительное длинное целое числовое значение, найденное в строке поиска. Функция игнорирует все ведущие нецифровые символы, кроме знака «минус» (-). Признаком конца целого числа служит первый нецифровой символ, следующий за числом. Если не обнаружено ни одного цифрового символа, функция возвращает значение 0.

- `long parseInt(char skipChar);`

  Практически такая же функция, как и функция `parseInt()`, но игнорирует символы, указанные в параметре `skipChar`. Эта возможность может быть полезной при парсинге большого числового значения, разбитого символами запятой на группы цифр. Но при этом следует иметь в виду, что эту функцию нельзя использовать для парсинга текстовых значений, разделенных запятыми. Например, подстроку `32,767` функция возвратит как `32786`.

- `float parseFloat();`

  Практически такая же функция, как и функция `parseInt()`, но для парсинга чисел с плавающей точкой (запятой). Функция игнорирует все нецифровые символы, кроме символа десятичной точки (`.`) и знака «минус» (-).

- `size_t readBytes(char *buffer, size_t length);`

  Считывает символы из потока в буфер до тех пор, пока не будет считано заданное количество символов или превышено время тайм-аута. Возвращает значение количества символов, помещенных в буфер.

- `size_t readBytesUntil(char terminator, char *buf, size_t length);`

  Считывает символы из потока в буфер до тех пор, пока не будет обнаружен символ разделителя (`terminator`), считано заданное количество символов или превышено время тайм-аута. Строки, превышающие размер буфера, усекаются, чтобы поместить их в буфер. Возвращает значение количества символов, помещенных в буфер.

Дополнительная информация

В *главе 15* приводятся дополнительные примеры парсинга потоковых данных для обнаружения и извлечения данных. Для парсинга на языке Arduino сообщений

в формате JSON можно использовать библиотеку ArduinoJson (**https://arduinojson.org**). Парсинг с использованием этой библиотеки рассматривается в *разд. 4.4*.

4.6. Отправка двоичных данных с платы Arduino

ЗАДАЧА

По какой-либо причине данные с платы Arduino необходимо передать в двоичном формате. Например, для передачи информации требуется использовать минимальное количество байтов, или же принимающее приложение может работать только с двоичными данными.

РЕШЕНИЕ

Эта задача решается скетчем, код которого приводится в листинге 4.13. Скетч отправляет сообщение, состоящее из заголовка, за которым следует двухбайтное целочисленное значение в двоичном формате. Целочисленное значение имеет тип `short`, поскольку независимо от разрядности используемой платы (8-разрядная или 32-разрядная), оно передается, как два отдельных байта. Значение генерируется с помощью функции языка Arduino `random()` (см. *разд. 3.11*). Хотя функция `random()` возвращает длинное целочисленное значение (типа `long`), она никогда не возвратит значение, большее, чем указанный здесь в параметре верхний предел 599. А такое значение поместится в целочисленную переменную типа `short`.

Листинг 4.13. Передача данных с платы Arduino в двоичном формате

```
/*
* Скетч SendBinary
* Отправляет сообщение, состоящее из заголовка, за которым следует
  произвольное двухбайтное значение в двоичном формате.
*/
short intValue; // Короткое (short) целочисленное значение
                // (два байта на всех платах)

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

void loop()
{
   Serial.print('H'); // Отправляем символ заголовка
   // Отправляем произвольное целое число
   intValue = random(599); // Генерируем произвольное целое число
                           // в диапазоне от 0 до 599
```

```
    // Отправляем два байта, из которых состоит это целое число
    Serial.write(lowByte(intValue));  // Отправляем младший байт
    Serial.write(highByte(intValue)); // Отправляем старший байт
    // Отправляем другое произвольное целое число
    intValue = random(599); // Генерируем произвольное целое число
                            // в диапазоне от 0 до 599
    // Отправляем два байта, из которых состоит данное целое число
    Serial.write(lowByte(intValue));  // Отправляем младший байт
    Serial.write(highByte(intValue)); // Отправляем старший байт
    delay(1000);
}
```

Обсуждение работы решения и возможных проблем

Задача передачи двоичных данных требует внимательного планирования, чтобы обе стороны согласились на такой способ передачи данных, ибо в противном случае принимающая сторона не сможет должным образом интерпретировать полученные данные. В отличие от текстовых данных, где конец сообщения можно обозначить разделительным символом возврата каретки (или каким-либо другим символом), определение начала и конца сообщения в двоичном формате для принимающей стороны может стать более трудной задачей, поскольку такие данные могут содержать такое же значение, что и заголовок или разделитель сообщения.

Эту проблему можно решить, согласовав между передающей и принимающей сторонами количество байтов, которого следует ожидать. При этом конец сообщения будет обозначаться количеством переданных байтов, а не нахождением определенного символа. Реализовать этот подход можно, указывая в первом передаваемом значении количество байтов, которые будут следовать после него. Или же можно использовать сообщения фиксированного размера — достаточно большого, чтобы в него помещались все данные, которые требуется передать. Любой из этих подходов не всегда легко реализовать на практике, поскольку в разных платформах и языках программирования могут использоваться разные размеры для данных двоичного типа — как количество байтов, так и порядок их передачи могут отличаться от соответствующих параметров, используемых в Arduino. Например, для 8-разрядных платформ Arduino целочисленный тип данных `int` определен как два байта (16 битов), но как четыре байта (32 бита) — для 32-разрядных. А в языке Processing (на основе Java) тип данных `int` определен как четыре байта (двухбайтовое целое число в Java определено типом `short`). Передача значения типа `int` в виде текста (как рассматривалось в предыдущих примерах) упрощает эту задачу, поскольку каждая отдельная цифра числа передается последовательно, т. е. в том порядке, в каком цифры расположены во всем передаваемом числе. Принимающая сторона узнает о полной передаче значения по приему символа возврата каретки или другого нецифрового разделяющего символа. В случае двоичного обмена принимающая сторона может знать о содержимом сообщения лишь в том случае, если оно определено наперед или указывается в самом сообщении.

Решение рассматриваемого в этом разделе примера требует согласования типа данных между сторонами обмена и обдуманного подхода к реализации обмена. В *разд. 4.7* приводится скетч на языке Processing для приема таких сообщений.

Передача одиночных байтов не представляет никаких проблем — для этого просто используем функцию: `Serial.write(byteValue)`. А для целочисленного значения (типа `int`) составляющие его младший и старший байты передаются по отдельности (дополнительная информация по типам данных приводится в *разд. 2.2*). Для передачи при этом используются функции `lowByte()` и `highByte()` (см. *разд. 3.14*):

```
Serial.write(lowByte(intValue));
Serial.write(highByte(intValue));
```

Передача четырех байтов длинного целочисленного значения (типа `long`) осуществляется в два этапа. Длинное значение сначала разбивается на два 16-битовые целые значения, каждое из которых затем передается с помощью только что рассмотренного метода для передачи двухбайтных целочисленных значений (типа `int`):

```
long longValue = 2147483648;
int intValue;
```

Сначала передается младшее 16-битовое целочисленное значение:

```
intValue = longValue & 0xFFFF; // Получаем значение младших 16 битов
Serial.write(lowByte(intValue));
Serial.write(highByte(intValue));
```

А затем передается старшее 16-битовое целочисленное значение:

```
intValue = longValue >> 16; // Получаем значение старших 16 битов
Serial.write(lowByte(intValue));
Serial.write(highByte(intValue));
```

Вместо того чтобы каждый раз, когда нужно передать целочисленное значение, использовать несколько строк кода, лучше сначала создать из этого кода соответствующую функцию, а затем вызывать ее по мере надобности. В листинге 4.14 приводится код функции для передачи 16-битового целого числа (типа `int`) по последовательному каналу связи.

Листинг 4.14. Функция для передачи целого числа типа `int`

```
// Функция для передачи целого числа типа int на последовательный порт

void sendBinary(int value)
{
   // Передаем побайтно двухбайтное целое число
   Serial.write(lowByte(value));  // Отправляем младший байт
   Serial.write(highByte(value)); // Отправляем старший байт
}
```

А в листинге 4.15 приводится рассмотренный ранее код для передачи по последовательному каналу связи четырехбайтного целого числа (типа `long`), организованный в функцию.

Листинг 4.15. Функция для передачи целого числа типа long

```
// Функция для передачи целого числа типа long на последовательный порт
void sendBinary(long value)
{
   // Сначала передаем младшее 16-битовое целочисленное значение
   int temp = value & 0xFFFF; // Получаем значение младших 16 битов
   sendBinary(temp);
   // А затем передаем старшее 16-битовое целочисленное значение
   temp = value >> 16; // Получаем значение старших 16 битов
   sendBinary(temp);
}
```

Функции в листингах 4.14 и 4.15 имеют одинаковое название: `sendBinary`. Компилятор различает эти две функции по типу передаваемого в параметре значения. Если при вызове функции `sendBinary()` ей передается двухбайтное значение, будет вызвана версия, объявленная как `sendBinary(int value)`. Если же в параметре передается четырехбайтное значение, то будет вызвана версия, объявленная как `sendBinary(long value)`. Это поведение называется *перегрузкой функции*. В скетче в листинге 4.2 приводился другой пример перегрузки функции. В частности, использование функции `Serial.print()` для передачи данных разных типов возможно благодаря тому, что компилятор различает разные варианты этой функции по типу передаваемого ей в параметре значения.

Двоичные данные также можно передавать по последовательному каналу связи, используя *структуры*. Структура представляет собой специальный механизм для организации данных. Но если вы не знакомы с использованием структур, то лучше воздержаться от их использования, а ориентироваться на рассмотренные ранее решения. Для тех же читателей, кому не страшно понятие указателей на структуру, в листинге 4.16 приводится код для передачи байтов двоичных данных по последовательному каналу связи. Это решение содержит символ заголовка в структуре, поэтому оно отправляет такое же сообщение, что и оригинальное решение из листинга 4.13.

Листинг 4.16. Передача двоичных данных с использованием структуры

```
/*
Скетч SendBinaryStruct
Передает структуру в виде двоичных данных
*/

typedef struct
{
   char padding; // Обеспечиваем одинаковое выравнивание
                 // для 8-битных и 32-битных платформ
   char header;
   short intValue1;
```

```
   short intValue2;
}
shortMsg;

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

void loop()
{
   shortMsg myStruct = { 0, 'H' ,(short) random(599), (short) random(599) };
   sendStructure((byte *)&myStruct, sizeof(myStruct));
   delay(1000);
}

void sendStructure(byte *structurePointer, int structureLength)
{
   int i;
   for (i = 0 ; i < structureLength ; i++)
   {
      Serial.write(structurePointer[i]);
   }
}
```

Если в объявлении структуры `shortMsg` не использовать элемент заполнителя `padding`, может оказаться, что на одних платах размер структуры будет пять байтов, а на других — шесть. Это объясняется тем, что компилятор для одной архитектуры может позволить пятибайтную структуру, тогда как компилятор для другой архитектуры может добавить один или два байта, чтобы размер структуры был кратным естественному размеру данных платы. Добавление заполнителя в начале структуры обеспечивает сохранение значения типа `char` по четному (второму) байту, что сделает маловероятным добавление компилятором заполнителя между этим значением и значениями типа `short`. Но маловероятное не означает невозможное, и не гарантирует, что этот трюк будет всегда срабатывать, поэтому может понадобиться поэкспериментировать. Еще одно достоинство добавления заполнителя перед символом заголовка состоит в том, что код, приведенный в *разд. 4.7*, будет игнорировать все входящие символы, пока не увидит символ английской буквы `H`.

Передача данных в двоичном формате более эффективна, чем в текстовом, но такая передача будет надежной только в том случае, когда состав данных полностью согласован между обеими сторонами обмена. Далее приводится краткое описание некоторых важных понятий, которым нужно уделять особое внимание при написании кода.

Размер переменной

Размер передаваемых данных должен быть одинаков на обеих сторонах обмена. Для плат Arduino Uno и других 8-разрядных плат размер целочисленного значения

типа int составляет два байта, тогда как для 32-разрядных плат и большинства других платформ размер этого типа данных составляет четыре байта. Поэтому всегда проверяйте размеры типов данных в документации по используемому языку программирования, чтобы обеспечить их сопоставимость. Прием 2-байтного целого числа (типа int) Arduino в 4-байтное целое число в скетче Processing не составляет никаких проблем при условии, что скетч Processing ожидает получить только два байта. Но при этом необходимо обеспечить, чтобы размер используемого передающей стороной типа данных не превышал размер соответствующего типа данных принимающей стороны, чтобы не допустить его переполнения.

Порядок байтов

Байты значений типа int или long должны передаваться в порядке, ожидаемом принимающей стороной. В рассматриваемом решении байты передаются в порядке, используемом платами Arduino для хранения данных. Этот порядок называется *прямым порядком*[4] (следования байтов), когда младший байт сохраняется по младшему адресу, а старший — по старшему. С технической точки зрения совместимые с Arduino платы с процессором ARM могут использовать как прямой, так и обратный порядок хранения байтов[5], но на практике вероятность встретить плату, использующую обратный порядок, ничтожно мала. Функции lowByte() и highByte(), применяемые для разборки целого числа на составляющие его байты, позволяют пользователю определять порядок передачи байтов. Но при передаче в двоичном формате данных, хранящихся в структуре, используется внутреннее представление структуры, которое определяется порядком хранения байтов для конкретной платы. Поэтому, если при приеме таких данных скетчем Processing для решения из *разд. 4.7* (код которого приводится в листинге 4.17) вы не получите ожидаемое значение (16384), возможно, что ваша структура использует обратный порядок хранения данных.

Синхронизация

Принимающая сторона должна быть способна определить начало и конец сообщения, поскольку если начать прием где-либо посредине сообщения, то полученные данные будут недействительными. Начало и конец сообщения можно обозначать последовательностью байтов, которая не будет встречаться нигде в собственно сообщении. Например, данные, возвращаемые функцией analogRead(), могут быть только в диапазоне от 0 до 1023, поэтому старший байт такого значения должен быть меньшим, чем 4. Это объясняется тем, что целочисленное значение (типа int) 1023 хранится в памяти как два байта: 3 и 255. Таким образом, данные никогда не будут содержать два последовательных байта больше чем 3. Соответственно, два последовательных байта со значением 4 (или любым другим значением больше чем 3) не могут быть действительными данными тела сообщения, и их можно использовать для обозначения начала или конца сообщения.

---

[4] В английской литературе — little endian.

[5] В английской литературе — big endian.

Управление потоком данных

Используйте такую скорость передачи, которая обеспечит принимающей стороне возможность успевать принимать их, или же примените управление потоком данных какого-либо типа. В управление потоком данных входят такие действия, как квитирование, посредством которого передающей стороне сообщается, что принимающая сторона готова снова принимать данные.

Дополнительная информация

Дополнительная информация по используемым в Arduino типам данных приводится в *главе 2*.

В *разд. 3.15* приводится дополнительная информация по работе со старшими и младшими байтами. На веб-сайте Arduino можно найти справочную информацию по функциям `lowByte()` и `highByte()` — **https://oreil.ly/AwmWi** и **https://oreil.ly/W0TJE** соответственно.

Дополнительная информация по управлению потоками данных приводится в Википедии (**https://oreil.ly/D4_2G**).

4.7. Прием на компьютере двоичных данных с платы Arduino

ЗАДАЧА

Необходимо создать программу на языке программирования типа Processing для приема на компьютере через последовательный порт двоичных данных, поступающих с платы Arduino, и последующей обработки этих данных. Такая программа могла бы, например, реагировать на сообщения, отправляемые скетчем Arduino из *разд. 4.6* (см. листинг 4.13).

РЕШЕНИЕ

Реализация этого решения зависит от среды программирования, установленной на вашем компьютере Windows или macOS. Для тех, кто еще не определился с платформой программирования, можно порекомендовать использовать язык Processing, который легко изучается и хорошо подходит для работы с платформой Arduino.

Следующие две строки кода на языке Processing из скетча SimpleRead (упоминаемого во врезке *«Среда разработки Processing»* в начале этой главы) считывают байт данных из последовательного порта:

```
if ( myPort.available() > 0) { // Если присутствуют данные,
val = myPort.read();          // считываем их и сохраняем в переменной val
```

Как можно видеть, этот код очень похож на код на языке Arduino, с которым мы работали в предыдущих решениях.

В листинге 4.17 приводится код скетча Processing, который принимает данные, посылаемые скетчем Arduino из *разд. 4.6*, и устанавливает размер прямоугольника пропорционально полученным целочисленным значениям.

Листинг 4.17. Прием и обработка двоичных данных, получаемых с платы Arduino

```
/*
* ReceiveBinaryData_P
*
* Значение переменной portIndex должно быть равно номеру порта COM,
  к которому подключена плата Arduino
*/
import processing.serial.*;

Serial myPort;  // Создаем экземпляр объекта класса Serial

// ВНИМАНИЕ!
// Если необходимо, измените указанный в следующей строке номер порта на правильный
short portIndex = 0; // Задаем номер порта COM. Нумерация портов
                     // начинается со значения 0

char HEADER = 'H';
int value1, value2; // Данные, получаемые на последовательный порт

void setup()
{
   size(600, 600);
   // Открываем последовательный порт, к которому подключена
   // плата Arduino
   String portName = Serial.list()[portIndex];
   println((Object[]) Serial.list());
   println(" Connecting to -> " + portName);
   myPort = new Serial(this, portName, 9600);
}

void draw()
{
   // Считываем заголовок и два двоичных *(16-разрядных)числа:
   if ( myPort.available() >= 5)   // Если доступны хотя бы 5 байтов
   {
      if( myPort.read() == HEADER) // если это заголовок
      {
         value1 = myPort.read();   // считываем младший байт
         value1 = myPort.read() * 256 + value1; // добавляем старший байт

         value2 = myPort.read();                // считываем младший байт
         value2 = myPort.read() * 256 + value2; // добавляем старший байт
```

```
        println("Message received: " + value1 + "," + value2);
      }
    }
    background(255); // Задаем белый цвет фона
    fill(0);         // Задаем черный цвет заливки

    // Прорисовываем прямоугольник по координатам на основе
    // целых чисел, полученных с платы Arduino
    rect(0, 0, value1,value2);
}
```

Значение переменной должно быть равным номеру порта COM, к которому подключена плата Arduino[6]. Если при первом исполнении скетча возникнет ошибка, проверьте список доступных портов в панели сообщений в нижней части окна Processing, чтобы определить, какой номер порта использовать для переменной `portIndex`.

Обсуждение работы решения и возможных проблем

Как можно видеть, скетчи Processing во многом похожи на скетчи Arduino. Эта схожесть преднамеренна, поскольку язык Arduino разработан на основе языка Processing. Точно так же, как и в скетчах Arduino, функция `setup()` в скетчах Processing используется для одноразовой инициализации. При исполнении скетчей Processing открывается окно, параметры и содержимое которого определяются исполняющимся скетчем. В нашем случае скетч задает размер окна в 600×600 пикселов, вызывая для этого функцию: `size(600,600)`.

Оператор:

```
String portName = Serial.list()[portIndex];
```

задает требуемый последовательный порт. В языке Processing все доступные последовательные порты перечисляются в объекте `Serial.list`. В нашем примере номер требуемого последовательного порта присваивается в качестве значения переменной `portIndex`.

Оператор:

```
println((Object[]) Serial.list());
```

отображает в окне консоли Processing список всех доступных последовательных портов, а оператор:

```
myPort = new Serial(this, portName, 9600);
```

открывает порт, указанный в переменной `portName`. При этом необходимо удостовериться, что переменной `portIndex` присвоено значение, соответствующее последова-

---

[6] При этом здесь и во всех аналогичных последующих случаях следует иметь в виду, что нумерация последовательных портов начинается со значения 0. Поэтому значение переменной `portIndex` для порта, например, COM2 будет 1, а не 2.

тельному порту, к которому подключена плата Arduino. В компьютерах macOS плата Arduino обычно подключается к первому последовательному порту, а на компьютерах Windows, оснащенных физическим последовательным портом RS-232, к последнему. Но при отсутствии на компьютере Windows физического порта подключение может быть осуществлено также к первому порту. Список доступных последовательных портов можно просмотреть в окне среды Arduino IDE (выполнив команду меню **Инструменты | Порт**). Здесь они могут быть перечислены в том же самом порядке, что и в окне среды Processing.

Функция `draw()` языка Processing функционирует подобно функции `loop()` языка Arduino, вызываясь бесконечно. В рассматриваемом скетче код функции `draw()` проверяет наличие данных в последовательном порту и при наличии таковых считывает их побайтно, преобразовывая байты в соответствующее целочисленное значение. Далее на основе полученных значений в окне скетча прорисовывается прямоугольник.

Дополнительная информация

Дополнительную информацию по языку программирования Processing можно найти на его веб-сайте (**http://processing.org**). Там же можно и загрузить установочный пакет.

4.8. Передача двоичных значений скетчем Processing на плату Arduino

ЗАДАЧА

Требуется передать целочисленные значения (типа `int` или `long`) в двоичном формате из скетча Processing на плату Arduino. Например, отправить сообщение, состоящее из идентифицирующего тега сообщения и двух 16-разрядных двоичных значений.

РЕШЕНИЕ

Эта задача решается скетчем Processing, код которого приводится в листинге 4.18.

Листинг 4.18. Передача целочисленных значений в двоичном формате на плату Arduino

```
// Скетч Processing

/* SendingBinaryToArduino
* Язык: Processing
*/
import processing.serial.*;

Serial myPort; // Создаем экземпляр объекта класса Serial
```

```
// ВНИМАНИЕ!
// Если необходимо, измените указанный в следующей строке номер порта на правильный
short portIndex = 0; // Задаем номер порта COM. Нумерация портов
                     // начинается со значения 0

public static final char HEADER = 'H';
public static final char MOUSE_TAG = 'M';

void setup()
{
   size(512, 512);
   String portName = Serial.list()[portIndex];
   println((Object[]) Serial.list());
   myPort = new Serial(this, portName, 9600);
}

void draw(
{
}

void serialEvent(Serial p)
{
   // Обрабатываем данные, поступающие по последовательному каналу связи
   String inString = myPort.readStringUntil('\n');
   if(inString != null)
   {
      print( inString ); // Отображаем полученную с Arduino текстовую строку
   }
}

void mousePressed()
{
   sendMessage(MOUSE_TAG, mouseX, mouseY);
}

void sendMessage(char tag, int x, int y
{
   // Отправляем этот указатель и значение на последовательный порт
   myPort.write(HEADER);
   myPort.write(tag);
   myPort.write((char)(x / 256)); // msb
   myPort.write(x & 0xff); //lsb
   myPort.write((char)(y / 256)); // msb
   myPort.write(y & 0xff); //lsb
}
```

Значение переменной должно быть равным номеру порта COM, к которому подключена плата Arduino. Если при первом исполнении скетча возникнет ошибка, проверьте список доступных портов в панели сообщений в нижней части окна Processing, чтобы определить, какой номер порта использовать для переменной `portIndex`.

Щелчок мышью в окне скетча Processing вызывает функцию `sendMessage()`, которой в качестве параметров передается 8-разрядный тег `MOUSE_TEG`, идентифицирующий сообщение, и два 16-разрядных значения координат X и Y щелчка мыши в окне скетча. Функция `sendMessage()` отправляет каждое из этих 16-разрядных значений побайтно, при этом старший байт отправляется первым.

В листинге 4.19 приводится код скетча Arduino для приема посылаемых ему скетчем Processing значений и отправки результата обратно этому скетчу.

Листинг 4.19. Прием значений, отправляемых скетчем Processing

```
// Скетч BinaryDataFromProcessing
// Эти определения должны быть идентичны соответствующим определениям
// в отправляющей программе
const char HEADER = 'H';
const char MOUSE_TAG = 'M';
const int TOTAL_BYTES = 6 ; // Общее количество байтов в сообщении

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

void loop(
{
   if ( Serial.available() >= TOTAL_BYTES)
   {
      if( Serial.read() == HEADER)
      {
         char tag = Serial.read();
         if(tag == MOUSE_TAG)
         {
            int x = Serial.read() * 256;
            x = x + Serial.read();
            int y = Serial.read() * 256;
            y = y + Serial.read();
            Serial.println("Got mouse msg:");
            Serial.print("x="); Serial.print(x);
            Serial.print(", y="); Serial.println(y);
         }
         else
         {
            Serial.print("Unknown tag: ");
```

```
            Serial.write(tag); Serial.println();
        }
    }
  }
}
```

Обсуждение работы решения и возможных проблем

Скетч Processing сначала отправляет байт заголовка `H`, указывающего, что за ним следует действительное сообщение. Передача заголовка `H` необходима для того, чтобы скетч Arduino мог синхронизировать прием, если он начнет работать посередине передаваемого сообщения или в случае потери данных в процессе передачи, что может случиться при использовании беспроводного канала связи. А тег `M` обеспечивает дополнительную проверку действительности сообщения, а также позволяет реализовать специализированную обработку возможных сообщений любых других типов. В рассматриваемом примере при вызове функции `sendMessage()` ей передаются три параметра: заголовочный тег и 16-разрядные значения координат X и Y щелчка мыши в окне скетча Processing.

На стороне приема скетч Arduino проверяет, что было получено как минимум количество байтов, указанное в переменной `TOTAL_BYTES`, обеспечивая тем самым, что обработка сообщения не начнется до тех пор, пока не будут получены все требуемые данные. После проверки на наличие заголовка и тега побайтно считываются два 16-разрядных значения, при этом первый байт умножается на 256, чтобы восстановить исходное значение старшего байта.

Получаемые скетчем Arduino данные можно отправлять по последовательному каналу связи не обратно скетчу Processing, а другому устройству — например, на ЖКД. Для подключения этого устройства нужно использовать программный последовательный порт, созданный с помощью библиотеки SoftwareSerial, или же дополнительный аппаратный последовательный порт платы, если она оснащена таковым. Соответствующий скетч приводится в *разд. 4.11* (см. листинг 4.26). Этот последовательный порт нужно будет инициализировать в функции `setup()` скетча Arduino, а также модифицировать все операторы `Serial.write()` и `Serial.print/println()` этого скетча для передачи в такой порт. Далее приводятся модификации, которые нужно выполнить в исходном скетче для передачи полученных данных на контакт 1 (`Serial1TX`) плат Arduino WiFi Rev 2, Arduino Leonardo и большинство совместимых с Arduino плат с процессором ARM. Так, в функцию `setup()` нужно добавить следующий код:

```
Serial1.begin(9600);
```

А операторы `print/println/write` в конце функции `loop()` модифицировать следующим образом:

```
Serial1.println();
Serial1.println("Got mouse msg:");
Serial1.print("x="); Serial1.print(x);
Serial1.print(", y="); Serial1.print(y);
```

и:

```
Serial1.println();
Serial1.print("Unknown tag: ");
Serial1.write(tag); Serial1.println();
```

Для правильной обработки двоичных сообщений передающая и принимающая стороны должны использовать одинаковый размер сообщения. Если увеличить или уменьшить размер передаваемого сообщения, значение переменной `TOTAL_BYTES` в скетче Arduino необходимо изменить соответствующим образом.

4.9. Передача выходных значений нескольких контактов Arduino

ЗАДАЧА

Требуется передавать из скетча Arduino группы целочисленных значений (типа `int` или `long`) в двоичном формате. Например, отправлять входные значения цифровых и аналоговых контактов платы Arduino скетчу Processing.

РЕШЕНИЕ

Скетч для решения этой задачи приводится в листинге 4.20. Скетч отправляет заголовок, за которым следует целое число, содержащее логические значения цифровых контактов со 2-го по 13-й, а за ним еще шесть целых чисел, содержащих значения аналоговых контактов с 0-го по 5-й. *Глава 5* содержит много решений, задающих входные значения цифровых и аналоговых контактов, которые можно использовать для проверки этого скетча.

Листинг 4.20. Передача логических значений нескольких контактов платы Arduino

```
/*
* Скетч SendBinaryFields
* Передает по последовательному каналу связи логические значения
  цифровых и аналоговых контактов в двоичном формате
*/

const char HEADER = 'H'; // Односимвольный заголовок, обозначающий
                         // начало сообщения

void setup()
{
   Serial.begin(9600);
   for(int i=2; i <= 13; i++)
   {
      pinMode(i, INPUT);     // Задаем входной режим работы
                             // для контактов со 2-го по 13-й
```

```
      digitalWrite(i, HIGH); // Включаем внутренние повышающие резисторы
   }
}

void loop()
{
   Serial.write(HEADER); // Отправляем заголовок
   // Переменные для значений битов и составленного из них целого числа
   int values = 0;
   int bit = 0;

   for(int i=2; i <= 13; i++)
   {
      bitWrite(values, bit, digitalRead(i)); // Присваиваем биту
      // значение 0 или 1 в зависимости от входного уровня
      // на соответствующем контакте
      bit = bit + 1; // Переходим к следующему биту
   }
   sendBinary(values); // Отправляем целое число

   for(int i=0; i < 6; i++)
   {
      values = analogRead(i);
      sendBinary(values); // Отправляем целое число
   }
   delay(1000); // Отправляем сообщения каждую секунду
}

// Функция для передачи целого числа на последовательный порт
void sendBinary(int value)
{
   // Отправляем два байта, из которых состоит наше целое число
   Serial.write(lowByte(value));  // Отправляем младший байт
   Serial.write(highByte(value)); // Отправляем старший байт
}
```

Обсуждение работы решения и возможных проблем

Код скетча сначала отправляет заголовок (символ `H`), за которым следует целое число, составленное с помощью функции `bitRead()` из логических значений входных уровней цифровых контактов. Затем отправляются шесть целых чисел, содержащие значения входных сигналов шести аналоговых контактов (дополнительная информация на этот счет приводится в *главе 5*). Все целочисленные значения передаются с помощью функции `sendBinary()`, с которой мы познакомились в *разд. 4.6*. Сообщение состоит из 15 байтов: один байт для символа заголовка, два байта для значений цифровых контактов и 12 байтов для значений аналоговых контактов.

Код для подачи входных сигналов на цифровые и аналоговые контакты платы Arduino рассматривается в *главе 5*.

Предположим, что на аналоговых контактах с 0-го по 5-й присутствуют входные значения 0, 100, 200, 200, 200, 200. Соответственно, на цифровых контактах со 2-го по 7-й присутствуют высокие входные уровни, а на цифровых контактах с 8-го по 13-й — низкие. Этим входным сигналам будут соответствовать значения байтов, составленные, как показано в листинге 4.21.

Листинг 4.21. Значения байтов, составленные из логических значений входных сигналов на аналоговых и цифровых контактах платы Arduino

```
72  // Символ H заголовка
    // Два байта в порядке младший-старший, содержащие биты,
    // представляющие контакты со 2-го по 13-й
63  // Двоичное значение 00111111, обозначающее высокие входные уровни
    // на контактах со 2-го по 7-й
0   // Означает низкие входные уровни на контактах с 8-го по 13-й

    // Два байта для каждого контакта, представляющие аналоговое значение
0   // Аналоговое значение контакта 0 равно 0 и отправляется
    // двумя байтами
0

100 // Значение контакта 1 равно 100 и отправляется двумя байтами
    // со значениями 100 и 0
0
...
    // Значения контакта 5 равно 500
244 // Остаток деления значения 500 на 256
1   // Сколько раз число 500 можно разделить на 256
2
```

В листинге 4.22 приводится код скетча Processing, который принимает передаваемое скетчем Arduino сообщение и выводит содержащиеся в нем значения в консоль окна Processing.

Листинг 4.22. Скетч Processing для приема сообщения Arduino со значениями, представляющими входные сигналы на его контактах

```
// Скетч Processing

/*
* ReceiveMultipleFieldsBinary_P
*
* Значение переменной portIndex должно быть равно номеру порта COM,
  к которому подключена плата Arduino
*/

import processing.serial.*;
```

```
Serial myPort;       // Создаем экземпляр объекта класса Serial
short portIndex = 0; // Задаем номер порта COM. Нумерация портов
                     // начинается со значения 0

char HEADER = 'H';

void setup()
{
   size(200, 200);
   // Открываем последовательный порт, к которому подключена плата Arduino
   String portName = Serial.list()[portIndex];
   println((Object[]) Serial.list());
   println(" Connecting to -> " + portName);
   myPort = new Serial(this, portName, 9600);
}

void draw()
{
   int val;

   if ( myPort.available() >= 15) // Ожидаем поступления всего сообщения
   {
      if( myPort.read() == HEADER) // Если это заголовок
      {
         println("Message received:");
         // Найден заголовок
         // Получаем целое число, содержащее значения битов
         val = readArduinoInt();
         // Отображаем значение каждого бита
         for(int pin=2, bit=1; pin <= 13; pin++)
         {
            print("digital pin " + pin + " = " );
            int isSet = (val & bit);
            if( isSet == 0)
            {
               println("0");
            }
            else
            {
               println("1");
            }
            bit = bit * 2; // Сдвигаем бит в следующую старшую двоичную позицию
         }
         println();
         // Выводим на экран шесть аналоговых значений
         for(int i=0; i < 6; i ++)
         {
            val = readArduinoInt();
```

```
            println("analog port " + i + "=" + val);
        }
        println("----");
    }
  }
}

// Возвращаем целочисленное значение байтов, полученных через
// последовательный порт (в порядке сначала младший, затем старший)
int readArduinoInt()
{
  int val; // Данные, получаемые на последовательный порт

  val = myPort.read();              // Считываем младший байт
  val = myPort.read() * 256 + val; // Добавляем старший байт
  return val;
}
```

Значение переменной должно быть равным номеру порта COM, к которому подключена плата Arduino. Если при первом исполнении скетча возникнет ошибка, проверьте список доступных портов в панели сообщений в нижней части окна Processing, чтобы определить, какой номер порта использовать для переменной `portIndex`.

Код скетча Processing ждет, пока не будут получены 15 символов. Если первый из полученных символов является заголовком, вызывается функция `readArduinoInt()`, которая считывает два байта и преобразовывает их обратно в целое число, выполняя математическую операцию, обратную операции, выполненной скетчем Arduino, чтобы получить отдельные биты, представляющие логические уровни цифровых контактов. Тогда шесть целых чисел представляют аналоговые значения. Обратите внимание на то, что по умолчанию логическое значение цифровых контактов будет равно 1 (высокий уровень). Это объясняется тем, что в скетче в листинге 4.20 мы включили их повышающие резисторы. Соответственно, если к такому контакту подключить кнопку, то логическое значение контакта при ненажатой кнопке будет 1, а при нажатой — 0. Эта тема рассматривается более подробно в *разд. 2.4*.

Дополнительная информация

Отправлять на компьютер значения контактов платы Arduino или задавать входные уровни на них с компьютера (не принимая решений на плате) можно с помощью тестовой программы Firmata (**http://www.firmata.org**). Библиотека Firmata и примеры скетчей входят в состав стандартного пакета установки среды Arduino IDE и могут быть вызваны командой меню **Файл | Примеры | Firmata**. Кроме того, на веб-сайте депозитория Github для Firmata по адресу: **https://github.com/firmata/arduino#firmata-client-libraries** можно загрузить клиентскую библиотеку для среды Processing (**https://github.com/firmata/processing**). Скетч Firmata загружаем в плату Arduino, с компьютера задаем входной или выходной режим работы кон-

тактов ввода/вывода платы, а затем устанавливаем или считываем состояние этих контактов.

4.10. Сохранение отправляемых Arduino данных в файл на компьютере

ЗАДАЧА

Сохранить данные, отправляемые скетчем Arduino по последовательному каналу связи, в файл на компьютере. Например, периодически сохранять в файл выходные значения цифровых и аналоговых контактов.

РЕШЕНИЕ

Решение задачи передачи информации с платы Arduino на компьютер мы рассмотрели в предыдущих разделах. Здесь для этого используется скетч Arduino из *разд. 4.9* (см. листинг 4.20). Скетч Processing, который занимается сохранением получаемых данных в файл, приводится в листинге 4.23. Этот скетч основан на скетче Processing, также рассмотренном в *разд. 4.9* (см. листинг 4.22).

Следующий скетч Processing (листинг 4.23) создает файл журнала (название которого состоит из текущей даты и времени) для сохранения получаемых данных в той же папке, в которой находится файл скетча. Сообщения, получаемые от скетча Arduino, добавляются в этот файл. Нажатие любой клавиши сохраняет файл и завершает работу программы.

Листинг 4.23. Скетч Processing для сохранения получаемых от Arduino данных в файл

```
/*
* Скетч ReceiveMultipleFieldsBinaryToFile_P
*
* Значение переменной portIndex должно быть равно номеру порта COM,
  к которому подключена плата Arduino
* Скетч сохраняет полученные данные в файл. Он основан на скетче
  ReceiveMultipleFieldsBinary
* Нажмите любую клавишу, чтобы завершить работу и сохранить файл
*/

import processing.serial.*;
import java.util.*;
import java.text.*;

PrintWriter output;
DateFormat fnameFormat = new SimpleDateFormat("yyMMdd_HHmm");
DateFormat timeFormat = new SimpleDateFormat("hh:mm:ss");
String fileName;
```

```
Serial myPort;       // Создаем экземпляр объекта класса Serial
short portIndex = 0; // Задаем номер порта COM. Нумерация портов
                     // начинается со значения 0
char HEADER = 'H';

void setup()
{
   size(200, 200);
   // Открываем последовательный порт, к которому подключена плата Arduino
   String portName = Serial.list()[portIndex];
   println((Object[]) Serial.list());
   println(" Connecting to -> " + portName);
   myPort = new Serial(this, portName, 9600);
   Date now = new Date();
   fileName = fnameFormat.format(now);
   output = createWriter(fileName + ".txt"); // Сохраняем файл журнала
                                             // в папке этого скетча
}

void draw()
{
   int val;
   if ( myPort.available() >= 15) // Ожидаем поступления всего сообщения
   {
      if( myPort.read() == HEADER) // Если это заголовок,
      {
         String timeString = timeFormat.format(new Date());
         println("Message received at " + timeString);
         output.println(timeString);

         // получаем целое число, содержащее значения битов
         val = readArduinoInt();
         // Отображаем значение каждого бита
         for (int pin=2, bit=1; pin <= 13; pin++)
         {
            print("digital pin " + pin + " = " );
            output.print("digital pin " + pin + " = " );
            int isSet = (val & bit);
            if (isSet == 0)
            {
               println("0");
               output.println("0");
            }
            else
            {
               println("1");
               output.println("1");
            }
```

```
            bit = bit * 2; // Сдвигаем бит
         }
         // Выводим на экран шесть аналоговых значений
         for (int i=0; i < 6; i ++)
         {
            val = readArduinoInt();
            println("analog port " + i + "=" + val);
            output.println("analog port " + i + "=" + val);
         }
         println("----");
         output.println("----");
      }
   }
}

void keyPressed()
{
   output.flush(); // Записываем оставшиеся данные в файл
   output.close(); // Закрываем файл
   exit();         // Завершаем исполнение программы
}

// Возвращаем целочисленное значение байтов, полученных
// через последовательный порт (в порядке сначала младший, затем старший)
int readArduinoInt()
{
   int val;             // Данные, получаемые на последовательный порт
   val = myPort.read(); // Считываем младший байт
   val = myPort.read() * 256 + val; // Добавляем старший байт
   return val;
}
```

Не забудьте присвоить переменной `portIndex` значение, соответствующее последовательному порту, к которому подключена плата Arduino. Если этой переменной присвоено неправильное значение, просмотрите вывод скетча Processing в консоли этой программы и в списке доступных последовательных портов выберите правильный порт.

Обсуждение работы решения и возможных проблем

Собственно название файла журнала создается с помощью функции Processing `DateFormat()`:

```
DateFormat fnameFormat= new SimpleDateFormat("yyMMdd_HHmm");
```

А полное имя файла создается добавлением к базовому названию расширения файла:

```
output = createWriter(fileName + ".txt");
```

Чтобы создать файл и завершить работу скетча, нужно нажать любую клавишу на клавиатуре при активном главном окне среды Processing. Единственное, что нельзя, — завершать работу скетча нажатием клавиши <Escape>, поскольку это завершит работу скетча без сохранения файла журнала. Файл журнала создается в той же самой папке, в которой находится файл скетча Processing (сам скетч нужно сохранить хотя бы один раз, чтобы создать эту папку). Определить папку скетча можно, выполнив команду меню среды Processing **Эскиз | Показать папку с эскизами**.

Функция Processing `createWriter()` открывает файл, в результате чего создается объект (единица функциональности времени исполнения) под названием `output`, который обрабатывает вывод данных в файл. В файл записывается тот же самый текст, который выводится в консоль окна Processing в *разд. 4.9*, но содержимое файла можно форматировать требуемым образом, используя для этого стандартные возможности Processing для работы со строками. Например, версия функции `draw()`, код которой показан в листинге 4.24, разделяет элементы содержимого файла запятыми, что позволяет считывать его в электронную таблицу или базу данных. Остальную часть кода скетча Processing можно оставить той же самой, хотя может быть полезным заменить расширение файла txt расширением csv.

Листинг 4.24. Форматирование содержимого файла разделяющими запятыми

```
void draw()
{
   int val;

   if (myPort.available() >= 15) // Ожидаем поступления всего сообщения
   {
      if (myPort.read() == HEADER) // Если это заголовок,
      {
         String timeString = timeFormat.format(new Date());
         output.print(timeString);
         val = readArduinoInt();
         // Отображаем значение каждого бита
         for (int pin=2, bit=1; pin <= 13; pin++)
         {
            int isSet = (val & bit);
            if (isSet == 0)
            {
               output.print(",0");
            }
            else
            {
               output.print(",1");
            }
            bit = bit * 2; // shift the bit
         }
```

```
        // Записываем шесть аналоговых значений, разделенных запятыми
        for (int i=0; i < 6; i ++)
        {
          val = readArduinoInt();
          output.print("," + val);
        }
        output.println();
      }
    }
}
```

Дополнительная информация

Дополнительная информация по функции `createWriter()` приводится на справочной странице веб-сайта Processing (**https://oreil.ly/1zHWE**). Язык Processing также содержит объект `Table`, позволяющий создавать, манипулировать и сохранять файлы CSV (**https://oreil.ly/9sqXb**).

4.11. Отправка данных с платы Arduino на несколько устройств с последовательным интерфейсом

ЗАДАЧА

Требуется передать данные по последовательному интерфейсу с платы Arduino на периферийное устройство — например, жидкокристаллический дисплей, но порт USB-Serial платы уже используется для взаимодействия с компьютером.

РЕШЕНИЕ

Для плат с несколькими аппаратными последовательными портами решение этой задачи не составляет никакой проблемы, что вкратце упоминается во *введении* в эту главу (*разд. 4.0*). Для этого нужно просто подключить контакты последовательного порта платы к соответствующим контактам периферийного устройства, как показано в схеме на рис. 4.5.

После чего инициализировать два последовательных порта и использовать порт `Serial` для взаимодействия с компьютером, а другой порт (обычно `Serial1`) — для взаимодействия с периферийным устройством. Соответствующий код приводится в листинге 4.25.

Листинг 4.25. Код для инициализации двух аппаратных последовательных портов платы

```
void setup()
{
  // Инициализируем два аппаратных порта платы
  Serial.begin(9600);   // Первый последовательный порт
```

```
Serial1.begin(9600);  // Некоторые платы имеют даже больше
                      // чем два последовательных порта
```

При использовании платы Arduino или совместимой платы с рабочим напряжением 3,3 В (например, платы с процессором SAMD), можно без опасений передавать данные на устройство с рабочим напряжением 5 В. Но если плата работает от напряжения 5 В (например, плата Arduino Uno), а периферийное устройство — от 3,3 В, то передача напрямую с платы на это устройство со временем повредит его. Такого развития событий можно избежать, используя делитель напряжения, чтобы понизить напряжение сигнала. Делители напряжения рассматриваются в разд. *4.13* и *5.11*.

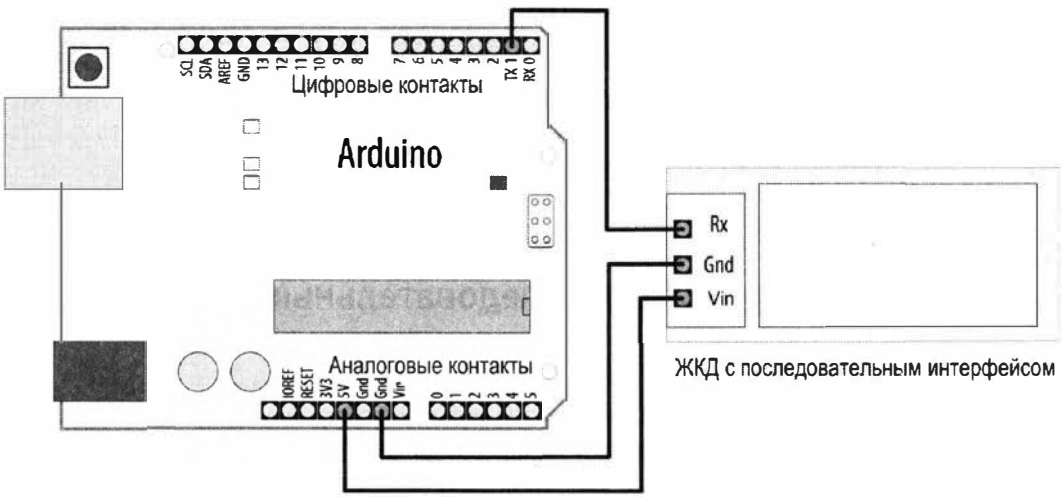


Рис. 4.5. Подключение периферийного устройства с последовательным интерфейсом к плате Arduino

В случае плат Arduino с процессором ATmega328 (например, Arduino Uno), оснащенных только одним аппаратным последовательным портом, нужно создать эмулированный, или программный, последовательный порт с помощью библиотеки SoftwareSerial.

Один из цифровых контактов платы выбирается в качестве передающего контакта этого порта, а другой — в качестве принимающего. Линия передачи устройства подключается к контакту приема платы, а линия приема — к контакту передачи. При однонаправленной передаче с платы на устройство, как в рассматриваемом случае передачи символов для отображения на ЖКД, подключать нужно только контакт передачи платы (TX) к контакту приема (RX) устройства. Такое подключение показано на рис. 4.6, где в качестве передающего контакта платы используется контакт 3.

Для работы с программным последовательным портом в скетче нужно создать экземпляр объекта класса `SoftwareSerial` и указать ему контакты платы, выбранные для этого порта. В следующем примере (листинг 4.26) мы создаем объект с названием `serial_lcd` и даем ему указание использовать для последовательного интерфейса контакты 2 и 3 платы Arduino. Хотя мы и не будем принимать данные через

этот последовательный порт, указать объекту `serial_lcd` контакт для приема все равно необходимо, поэтому при использовании этого программного порта контакт 2 нельзя использовать для каких бы то ни было других целей.

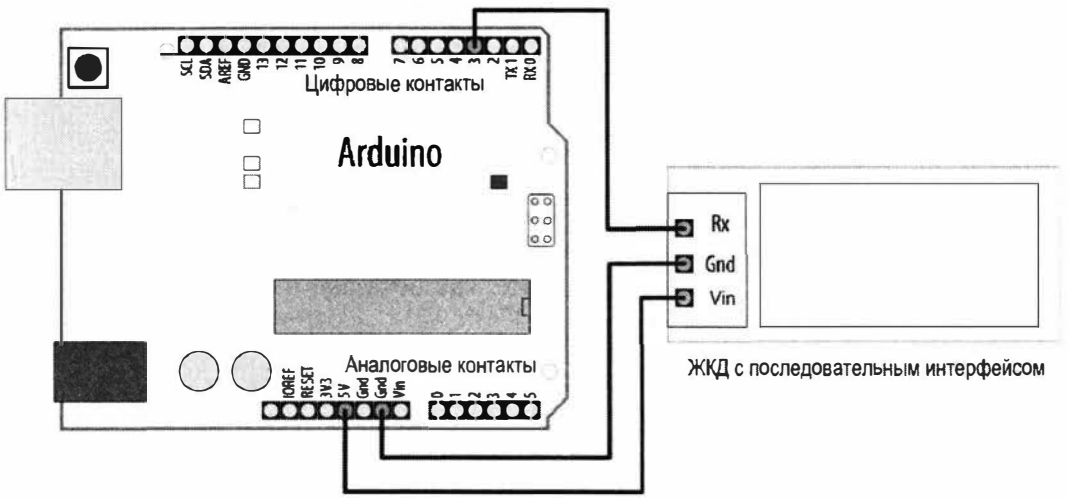


Рис. 4.6. Подключение периферийного устройства с последовательным интерфейсом к программному последовательному порту платы Arduino

Листинг 4.26. Передача данных по программному последовательному порту

```
/*
* Скетч SoftwareSerialOutput
* Передача данных через программный последовательный порт
*/

#include <SoftwareSerial.h>

const int rxpin = 2; // Контакт платы для приема данных (в этом примере
                     // не используется)
const int txpin = 3; // Контакт платы для передачи данных на ЖКД
SoftwareSerial serial_lcd(rxpin, txpin); // Программный последовательный
                                         // порт использует контакты 2 и 3

void setup()
{
   Serial.begin(9600);      // Для аппаратного порта задаем скорость передачи 9600 бод
   serial_lcd.begin(9600); // Для программного порта задаем
                           // такую же скорость передачи
}

int number = 0;

void loop()
```

```
{
   serial_lcd.print("Number: "); // Передаем текстовые данные на ЖКД
   serial_lcd.println(number);   // Передаем числовые данные на ЖКД
   Serial.print("Number: ");
   Serial.println(number);       // Передаем числовые данные на монитор порта
   delay(500);                   // Пауза в полсекунды перед выводом
                                 // следующего числа
   number++;                     // Переходим к выводу следующего числа
}
```

Выполнив небольшую модификацию, этот скетч можно использовать также и для передачи данных на ЖКД через дополнительный аппаратный порт платы Arduino. Для этого ЖКД нужно подключить к плате, как показано на рис. 4.5, а из скетча удалить следующий код:

```
#include <SoftwareSerial.h>
const int rxpin = 2;
const int txpin = 3;
SoftwareSerial serial_lcd(rxpin, txpin);
```

Затем на место удаленного кода вставить следующий оператор:

```
#define serial_gps Serial1
```

Если плата оснащена более чем двумя аппаратными последовательными портами, и используется не второй, а иной порт, вместо 1 в идентификаторе `Serial1` нужно указать соответствующее число.

Для некоторых плат, оснащенных несколькими аппаратными последовательными портами (например, Leonardo, Mega и Mega 2560), существуют ограничения на контакты, которые можно использовать для приема (RX) через программный последовательный порт. Хотя в нашем случае мы не осуществляем прием на плату Arduino, и в случае этих плат будет, скорее всего, использоваться аппаратный порт `Serial1` (см. табл. 4.1), следует иметь в виду, что эти платы не поддерживают прием (RX) для программного порта на контакте 2. Поэтому при организации в этих платах программного последовательного порта для его линии приема следует использовать разрешенный для этой цели контакт. В *разд. 4.12* (см. листинг 4.27) для линии приема (RX) программного последовательного порта используется контакт платы Arduino, который будет приемлем для широкого диапазона плат.

Для целей этого скетча предполагается, что ЖКД подключен к контакту 3 платы Arduino, как показано на рис. 4.6, и что монитор порта подключен к аппаратному последовательному порту. Код главного цикла `loop()` скетча будет бесконечно выводить одинаковое сообщение на каждом из них:

```
Number: 0
Number: 1
Number: 2
Number: 3
...
```

Обсуждение работы решения и возможных проблем

Микроконтроллер платы Arduino оснащен как минимум одним встроенным аппаратным последовательным портом. В случае платы Arduino Uno линии этого порта подключены (через преобразователь USB-Serial) к разъему USB платы, а также к контактам 0 (RX — прием) и 1 (TX — передача), что позволяет подключать к плате периферийные устройства с последовательным интерфейсом — например, жидкокристаллический дисплей. Символы, передаваемые с помощью объекта `Serial`, отображаются на ЖКД.

Хотя для периферийного устройства можно использовать отдельный источник питания, шину «земли» этого устройства необходимо подключить к шине «земли» (контакт GND) платы Arduino, чтобы у них была общая «земля». В этом решении мы реализовали такое подключение, но также использовали выходное напряжение 5 В платы Arduino (контакт 5V) для питания жидкокристаллического дисплея.

Кроме аппаратного последовательного порта USB-Serial некоторые платы оснащены дополнительно одним или несколькими аппаратными портами. Контакты 0 и 1 таких плат обычно связаны с объектом `Serial1`, что позволяет одновременно поддерживать связь с компьютером по подключению USB и реализовать обмен данными с другим устройством, подключенным к этим контактам. При этом ряд других плат поддерживают дополнительные аппаратные последовательные порты на других контактах. В табл. 4.1 приводится информация о доступных аппаратных портах нескольких видов плат и используемых ими для этого контактов. Все контакты, которые кроме использования в качестве контактов общего ввода/вывода, можно также задействовать в качестве контактов приема/передачи аппаратного последовательного порта, поддерживаются встроенным в микроконтроллер аппаратным *универсальным асинхронным приемопередатчиком* (УАПП). Это аппаратное устройство отвечает за генерирование последовательности синхронизированных с высокой точностью импульсов, которые противоположное устройство интерпретирует как данные, а также за интерпретирование подобного потока импульсов, получаемых им от периферийного устройства.

Хотя платы, оснащенные процессорами SAMD архитектуры ARM (платы M0 и M4) оснащены двумя аппаратными последовательными портами, а плата Mega даже четырьмя, плата Arduino Uno и большинство подобных плат на микроконтроллере ATMega328 оснащены только одним аппаратным портом. Поэтому, если к плате Uno или подобной плате нужно подключить по последовательному интерфейсу больше одного устройства, требуемые для этого дополнительные последовательные порты необходимо реализовать программно с помощью соответствующей библиотеки SoftwareSerial. Эта библиотека фактически превращает произвольную пару цифровых контактов общего ввода/вывода в новый последовательный порт.

Чтобы создать программный последовательный порт, нужно выбрать пару контактов платы для работы в качестве линий приема и передачи во многом в том же порядке, как работают контакты приема/передачи аппаратного последовательного порта. На рис. 4.6 показаны используемые таким образом контакты 3 и 2, но с некоторыми исключениями для определенных плат (**https://oreil.ly/ewtoN**) в этих целях

можно задействовать любые доступные цифровые контакты ввода/вывода общего назначения. При этом будет разумным избегать использования для программного последовательного порта контактов 0 и 1 платы, поскольку они уже задействуются аппаратным последовательным портом.

Передача данных через программный последовательный порт осуществляется точно таким же образом, как и через аппаратный порт. В скетче решения (см. листинг 4.26) данные отправляются как через аппаратный, так и через программный последовательные порты, с помощью функций `print()` и `println()`:

```
serial_lcd.print("Number: "); // Передаем текстовые данные на ЖКД
serial_lcd.println(number);   // Передаем числовые данные на ЖКД
Serial.print("Number: ");
Serial.println(number);       // Передаем числовые данные на монитор порта
```

Если общее количество символов передаваемого на ЖКД текста и чисел превышает количество позиций строки ЖКД, то лишние символы могут отсекаться или же выводимый текст может в строке дисплея прокручиваться. Большинство ЖК-дисплеев могут отображать две строки по 20 символов в каждой.

Дополнительная информация

Версия SendOnlySoftwareSerial библиотеки SoftwareSerial, разработанная Ником Гаммоном (Nick Gammon), поддерживает только передачу, что позволяет избежать выделения контакта для приема, когда этого не требуется. Библиотеку можно загрузить с ее депозитория GitHub (**https://oreil.ly/PBkw0**).

4.12. Прием данных платой Arduino по последовательному интерфейсу от нескольких периферийных устройств

ЗАДАЧА

Требуется принимать данные на плату Arduino по последовательному интерфейсу от периферийного устройства — например, модуля GPS для глобального позиционирования, но порт USB-Serial платы уже занят для взаимодействия с компьютером.

РЕШЕНИЕ

Для плат с несколькими аппаратными последовательными портами решение этой задачи не представляет никакой проблемы, что вкратце упоминается во *введении* в эту главу (*разд. 4.0*). Для этого нужно просто подключить контакты последовательного порта платы к соответствующим контактам периферийного устройства, как показано в схеме на рис. 4.7. После чего инициализировать два последовательных порта и использовать порт `Serial` для взаимодействия с компьютером, а другой

порт (обычно `Serial1`) — для взаимодействия с периферийным устройством. Соответствующий код приводится в листинге 4.27.

Листинг 4.27. Код для инициализации двух последовательных портов платы

```
void setup()
{
   // Инициализируем два аппаратных порта платы
   Serial.begin(9600);  // Первый последовательный порт
   Serial1.begin(9600); // Некоторые платы имеют даже больше
                        // чем два аппаратных последовательных порта
}
```

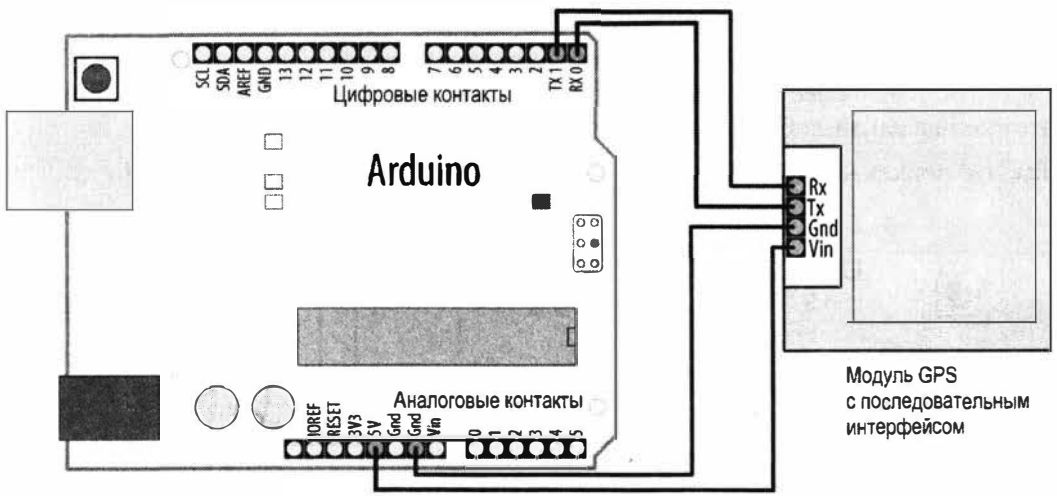


Рис. 4.7. Подключение периферийного устройства с последовательным интерфейсом к плате Arduino

При использовании платы Arduino или совместимой платы с рабочим напряжением 5 В (например, платы Uno), можно без опасений принимать данные с устройства с рабочим напряжением 3,3 В. Но если плата работает от напряжения 3,3 В (например, большинство плат с процессорами архитектуры ARM), а периферийное устройство использует логические уровни напряжением 5 В, то прием напрямую с такого устройство на плату со временем повредит плату. Такого развития событий можно избежать, используя делитель напряжения, чтобы понизить напряжение сигнала. Делители напряжения рассматриваются в *разд. 4.13* и *5.11*.

В случае плат Arduino с процессором ATmega328 (например, Arduino Uno), оснащенных только одним аппаратным последовательным портом, нужно создать эмулированный, или программный, последовательный порт с помощью библиотеки SoftwareSerial. При этом скорость обмена через программный последовательный порт будет меньше, чем через аппаратный порт.

Эта задача подобна задаче в *разд. 4.11*, и решение для нее, по сути, во многом то же самое. Если аппаратный порт платы используется для взаимодействия с монитором

порта, но к плате требуется подключить другое устройство с последовательным интерфейсом, необходимо создать программный последовательный порт с помощью библиотеки SoftwareSerial. В нашем случае этот порт будет использоваться не для передачи данных с платы, а для приема их платой, но базовое решение во многом такое же.

Для этого нужно выбрать два цифровых контакта ввода/вывода общего назначения для использования в качестве линий передачи и приема. В рассматриваемом решении для этой цели взяты контакты 8 и 9, поскольку некоторые платы (например, Arduino Leonardo) могут поддерживать прием через программный порт только на контактах 8, 9, 10, 11 и 14. Эти же контакты поддерживаются для приема через программный порт платами Arduino Mega и Mega 2560. Поскольку платы Mega оснащены двумя дополнительными аппаратными портами (`Serial2` и `Serial3`), то на практике, скорее всего, для подключения периферийного устройства к ним использовались бы контакты 0 и 1 аппаратного порта `Serial1`. Но здесь мы решили выбрать для программного последовательного порта такие контакты, которые будут работать с наиболее широким диапазоном плат, если вы решите протестировать этот код на какой-либо из них.

Так что подключите модуль GPS к плате Arduino, как показано на рис. 4.8.

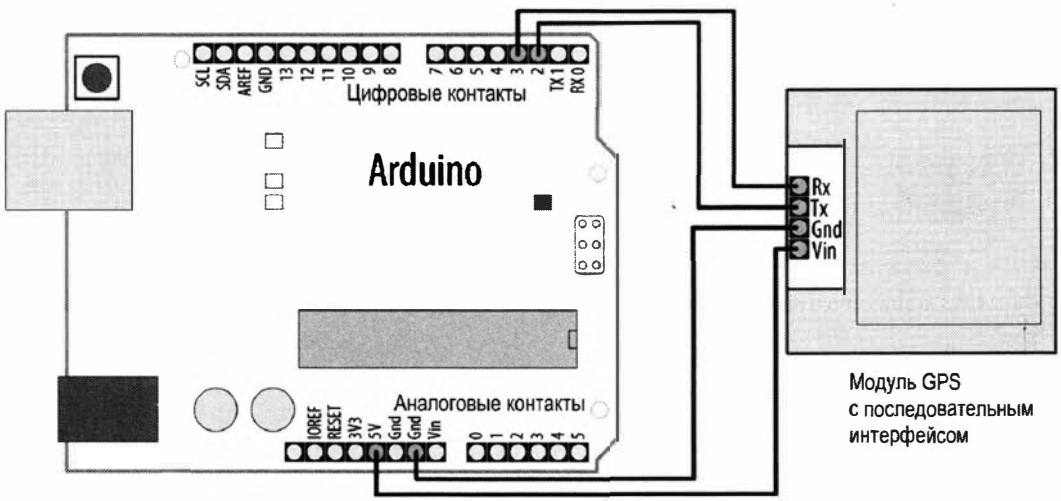


Рис. 4.8. Подключение модуля GPS с последовательным интерфейсом к программному последовательному порту платы Arduino

Подобно решению, приведенному в *разд. 4.11*, в скетче нужно создать экземпляр объекта класса `SoftwareSerial` и указать ему контакты платы, выбранные для этого порта. В скетче решения, код которого приводится в листинге 4.28, мы определяем программный последовательный порт с названием `serial_gps`, используя в нем для приема и передачи контакты 8 и 9 соответственно. Хотя мы и не будем отправлять данные на модуль GPS, для объекта `serial_gps` все равно необходимо указать контакт для передачи данных, поэтому при использовании этого программного порта контакт 9 нельзя использовать ни для каких других целей.

Выполнив небольшую модификацию, этот скетч (см. листинг 4.28) можно использовать также для работы с дополнительным аппаратным портом платы Arduino. Для этого модуль GPS нужно подключить к плате, как показано на рис. 4.7, из скетча удалить следующий код:

```
#include <SoftwareSerial.h>
const int rxpin = 8;
const int txpin = 9;
SoftwareSerial serial_gps(rxpin, txpin);
```

а затем на место удаленного кода нужно вставить следующий оператор:

```
#define serial_gps Serial1
```

Если плата оснащена более чем двумя аппаратными последовательными портами и используется не второй, а иной порт, вместо 1 в идентификаторе `Serial1` нужно указать соответствующее число.

Листинг 4.28. Скетч для приема данных от модуля GPS по программному порту

```
/*
* Скетч SoftwareSerialInput
* Прием данных через программный последовательный порт
*/

#include <SoftwareSerial.h>
const int rxpin = 8; // Контакт платы для приема данных от модуля GPS
const int txpin = 9; // Контакт платы для передачи данных на модуль GPS
SoftwareSerial serial_gps(rxpin, txpin); // Программный последовательный
                                         // порт использует контакты 8 и 9

void setup()
{
   Serial.begin(9600); // Для аппаратного порта задаем скорость передачи 9600 бод
   serial_gps.begin(9600); // Инициализируем программный
                  // последовательный порт. Большинство устройств GPS
                  // используют скорость 9600 бод
}

void loop()
{
   if (serial_gps.available() > 0) // Проверяем наличие входящих данных
   {
      char c = serial_gps.read(); // При наличии таковых, считываем их
      Serial.write(c);            // и отображаем в окне монитора порта
   }
}
```

Этот короткий скетч просто пересылает на монитор порта все данные, которые он получает от модуля GPS. Если модуль GPS исправен и правильно подключен

к плате Arduino, в окне монитора порта должны выводиться данные, отправляемые этим модулем.

Обсуждение работы решения и возможных проблем

Инициализация программного последовательного порта осуществляется объявлением контактов платы Arduino для приема и передачи данных. Следующий код задает контакты 8 и 9 для приема и передачи соответственно:

```
const int rxpin = 8;    // Контакт платы для приема данных от модуля GPS
const int txpin = 9;    // Контакт платы для передачи данных на модуль GPS
SoftwareSerial serial_gps(rxpin, txpin); // Программный
                                  // последовательный порт использует контакты 8 и 9
```

Код для приема данных через программный последовательный порт очень похож на код для приема данных через аппаратный порт. Сначала посредством функции `available()` проверяем наличие входящих символов и при положительном результате проверки считываем их с помощью функции `read()`.

Важно иметь в виду, что программные последовательные порты работают более медленно и потребляют больше вычислительных ресурсов, чем аппаратные. Программный последовательный порт должен делать все, что делает аппаратный порт, занимая для этого ресурсы процессора, который используется скетчем для выполнения «настоящей» работы. По прибытии нового символа процессор должен бросить все, чем он занимается в текущий момент, чтобы обработать прием этого символа. Для этого может потребоваться некоторое время. Например, при скорости передачи 4800 бод обработка одного символа занимает у микроконтроллера платы Arduino около 2 мс. На первый взгляд, 2 мс могут показаться не таким уж и большим временем, но если вдуматься, то при скорости передачи 200–250 символов в секунду наш скетч будет уделять 40–50% своего рабочего времени только приему поступающих данных. В результате для обработки этих данных остается очень мало времени. Из этого следует рекомендация, что если по последовательному интерфейсу необходимо взаимодействовать с двумя устройствами, то по мере возможности устройство с бóльшим объемом данных обмена следует подключáть к встроенному аппаратному последовательному порту. Если же такое устройство приходится подключать через программный последовательный порт, то необходимо принять все меры для обеспечения эффективной работы остального кода скетча, не занятого приемом данных.

Прием данных по нескольким программным портам

Библиотека SoftwareSerial позволяет создать несколько программных последовательных портов в одном скетче. Эта возможность может быть полезной для, например, управления несколькими радиомодулями XBee (см. *разд. 14.2*) или несколькими дисплеями в одном проекте. Но, к сожалению, в любой момент времени принимать данные может только один из этих портов. Надежная связь через программный порт требует непрестанного внимания процессора, чем и объясняется то, что одновременно может быть задействован только один программный порт.

При этом один скетч может выполнять прием через два разных программных порта. Нужно только, чтобы прием на этих портах не выполнялся одновременно. Существует много успешных подходов для работы с двумя программными портами, когда в течение некоторого времени принимаются данные от, например, модуля GPS, а затем — от радиомодуля XBee. Ключевой момент здесь заключается в том, чтобы не спешить переключаться между устройствами, а переходить к другому устройству только по завершении приема от первого.

Например, скетч в листинге 4.29 принимает по радиомодулю XBee команды, передаваемые другим радиомодулем XBee, подключенным к удаленному устройству. Скетч принимает поток данных от радиомодуля XBee через один программный порт, пока не получит команду начать принимать данные от модуля GPS, подключенного к другому программному порту. Тогда скетч в течение 10 секунд принимает данные от модуля GPS, чего должно быть достаточно, чтобы определить местоположение, после чего возвращается к приему данных от модуля XBee.

Итак, при использовании нескольких программных портов данные одновременно принимаются только по одному из них. Такой порт называется *активным*. По умолчанию активным является порт, для которого был выполнен самый последний вызов функции `begin()`. Но сделать программный порт активным можно в любое время, вызвав его функцию `listen()`. Функция `listen()` дает указание системе SoftwareSerial прекратить прием данных на одном порту и начать делать это на другом.

Поскольку скетч в листинге 4.29 только принимает данные, для передачи через оба порта можно выбрать любые контакты платы Arduino (переменные `txpin1` и `txpin2`). Если контакты 9 и 11 нужны вам для каких-либо иных целей, то переменным `txpin1` и `txpin2` можно присвоить другие значения. При этом не рекомендуется использовать несуществующие номера контактов, поскольку это может вызвать непредсказуемое поведение скетча.

Следующий скетч (листинг 4.29) иллюстрирует, как может выглядеть код для приема данных сначала с одного программного порта, а затем с другого.

Листинг 4.29. Прием данных по двум программным последовательным портам

```
/*
* Скетч MultiRX
* Принимает данные по двум программным последовательным портам
*/

#include <SoftwareSerial.h>
const int rxpin1 = 8;
const int txpin1 = 9;
const int rxpin2 = 10;
const int txpin2 = 11;

SoftwareSerial gps(rxpin1, txpin1);  // Линию передачи (TX) модуля GPS
                                     // подключаем к контакту 9 платы Arduino
```

```
SoftwareSerial xbee(rxpin2, txpin2); // Линию передачи (TX) радиомодуля Xbee
                                     // подключаем к контакту 10 платы Arduino

void setup()
{
   Serial.begin(9600);
   xbee.begin(9600);
   gps.begin(9600);
   xbee.listen(); // Делаем активным порт радиомодуля XBee
}

void loop()
{
   if (xbee.available() > 0) // Активный порт устройства Xbee.
                             // Есть ли символы?
   {
      if (xbee.read() == 'y') // Если от модуля Xbee получили символ 'y',
      {
         gps.listen(); // начинаем прием на порту устройства GPS
         unsigned long start = millis();   // Ведем прием на порту устройства GPS
         while (start + 100000 > millis()) // в течение 10 секунд
         {
            if (gps.available() > 0) // Активный порт устройства GPS
            {
               char c = gps.read();
               Serial.write(c); // Отображаем полученные данные GPS
                                // в окне монитора порта
            }
         }
         xbee.listen(); // По истечении 10 секунд возвращаемся к приему
                        // на порту устройства XBee
      }
   }
}
```

Этот скетч организован таким образом, что порт радиомодуля XBee является активным портом, пока не будет получен символ `'y'`, после чего активным становится порт устройства GPS. После приема на порту в течение 10 секунд данных от устройства GPS скетч возвращается к прослушиванию порта радиомодуля XBee. Данные, прибывающие на неактивный порт, просто игнорируются.

Обратите внимание, что ограничение активного порта действительно только в случае использования нескольких программных последовательных портов. Поэтому, если для вашего проекта необходимо действительно одновременно вести обмен данными с несколькими устройствами, рассмотрите возможность применить плату, оснащенную несколькими аппаратными последовательными портами. Например, плату Teensy 4.0, которая оснащена четырьмя такими портами. В табл. 4.1 ранее

приводился список контактов цифрового ввода/вывода, используемых в разных платах Arduino и их клонах для последовательных портов.

Дополнительная информация

В *разд. 6.14* приводится дополнительная информация по работе с устройством GPS — в частности, как выполнять парсинг принимаемых данных.

Библиотека AltSoftSerial (**https://oreil.ly/-odVk**) предоставляет более эффективную поддержку нескольких программных портов, что делает ее хорошей альтернативой библиотеке SoftSerial.

4.13. Использование Arduino с Raspberry Pi

ЗАДАЧА

Необходимо расширить возможности платы Arduino вычислительной мощностью одноплатного компьютера типа Raspberry Pi. Например, отправлять плате Arduino команды из сценария, исполняющегося на Raspberry Pi.

РЕШЕНИЕ

С помощью скетча Arduino можно принимать и исполнять команды, издаваемые Raspberry Pi. Скетч в листинге 4.30 управляет подключенными к плате Arduino светодиодами, исполняя команды, издаваемые сценарием, исполняющимся на Raspberry Pi.

Вместо последовательного порта плату Arduino можно подключить к компьютеру Raspberry Pi через один из его разъемов USB. Более того, на нем можно даже установить и использовать версию среды разработки Arduino IDE для процессоров ARM (**https://oreil.ly/K5JhS**). На момент подготовки этой книги для Raspberry Pi использовалась 32-разрядная операционная система Raspbian, поэтому для загрузки следует выбрать эту версию, если только у вас не установлена 64-разрядная операционная система.

Подключите контакт приема последовательного порта платы Arduino (контакт 0, обозначенный RX) к контакту 8 платы Raspberry Pi, а контакт передачи платы Arduino (контакт 1, обозначенный TX) к контакту 10 платы Raspberry Pi. Затем подключите контакт «земли» платы Arduino (обозначенный GND) к одному из контактов «земли» платы Raspberry Pi (мы использовали контакт 14, как показано на рис. 4.9).

Скетч в листинге 4.30 предполагает использование платы Arduino Uno или совместимой с ним платы, оснащенной одним аппаратным последовательным портом, используемым совместно с разъемом USB и контактами RX/TX. Если используется плата с несколькими аппаратными последовательными портами — например, Leonardo, WiFi Rev2, Nano Every или любая плата с процессором ARM, замените оператор:

```
#define mySerial Serial
```

оператором:

```
#define mySerial Serial1
```

А если в вашей плате для линий RX/TX используются иные контакты, чем 0 и 1, укажите соответствующе контакты для порта `Serial1` (см. табл 4.1).

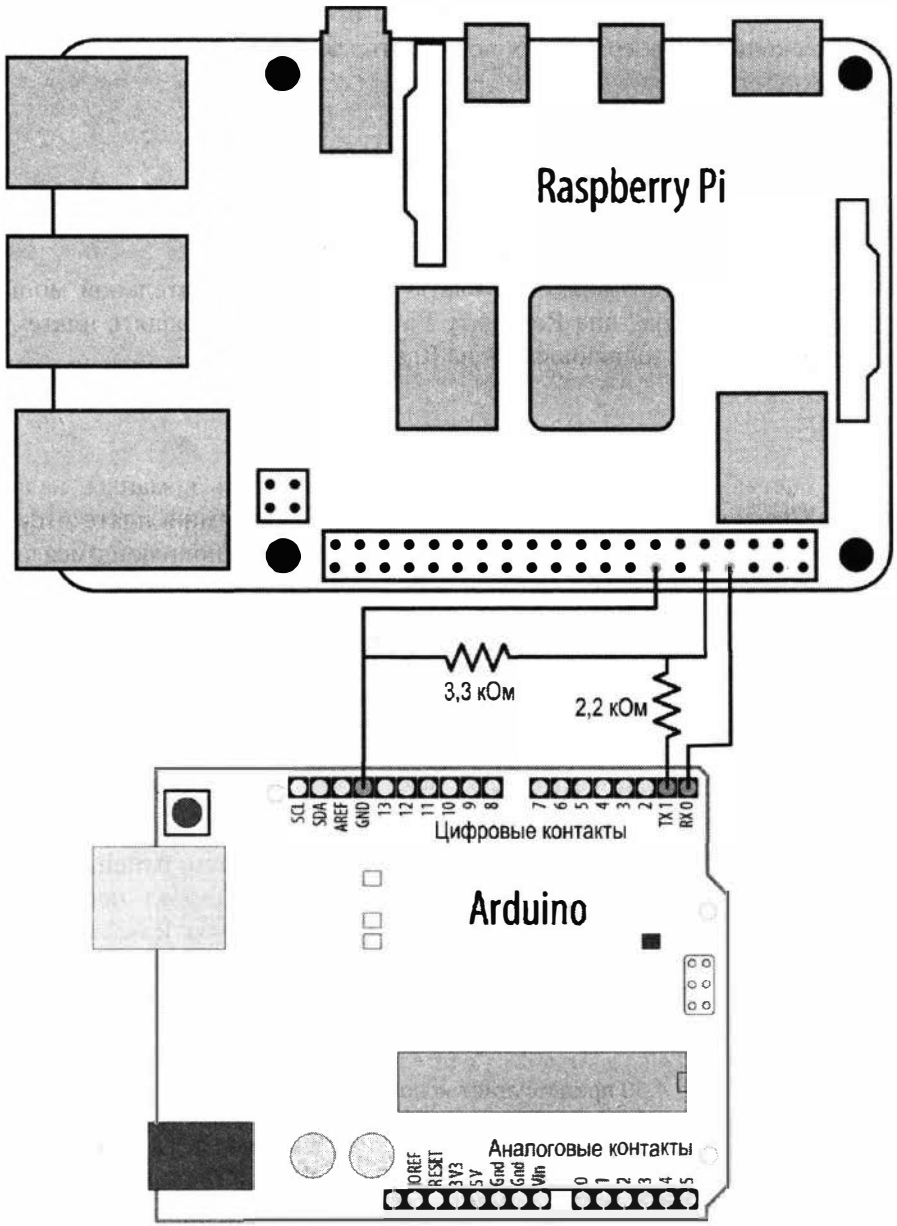


Рис. 4.9. Подключение платы Arduino к одноплатному компьютеру Raspberry Pi

Следующий скетч Arduino принимает по последовательному интерфейсу сообщения, отправляемые платой Raspberry Pi. Загрузите его в свою плату Arduino.

Листинг 4.30. Прием сообщений платой Arduino от компьютера Raspberry Pi

```
/*
* Скетч ArduinoPi
* Управления контактами Arduino посредством сообщений сценария
  Raspberry Pi, передаваемых по последовательному интерфейсу
* Формат сообщений: Pn = состояние
* где 'P" — символ заголовка, n — номер контакта, состояние = 0 или 1
* Пример: P13=1 включает контакт 13
*/

// Для плат с дополнительными аппаратными портами замените Serial на Serial1
#define mySerial Serial

void setup()
{
   mySerial.begin(9600); // Задаем скорость обмена по последовательному
                         // порту 9600 бод
}

void loop()
{
   if (mySerial.available()) // Проверяем наличие хотя бы одного принятого символа
   {
      char ch = mySerial.read();
      int pin = -1;
      if( ch == 'P') // Начало сообщения для установки состояния контакта?
      {
         pin = mySerial.parseInt(); // Получаем номер контакта
      }
      else if (ch == 'B') // Сообщение для включения встроенного светодиода
      {
         pin = LED_BUILTIN;
      }

      if( pin > 1)
      { // Контакты 0 и 1 обычно используются последовательным
        // интерфейсом (не используем их для управления светодиодами)
         int state = mySerial.parseInt(); // 0 — включить светодиод, 1 — выключить
         pinMode(pin, OUTPUT);
         digitalWrite(pin, state);
      }

   }
}
```

Сохраните сценарий Python из листинга 4.31 на своей плате Raspberry Pi под именем blinkArduino.py, запустите консоль командной строки, перейдите в папку, содержащую сценарий, и запустите его на исполнение, введя в консоли команду:

```
python blinkArduino.py
```

Исполняющийся сценарий будет передавать на плату Arduino команду для мигания ее встроенным светодиодом. Но прежде чем исполнять сценарий, на компьютере Raspberry Pi нужно установить библиотеку python-serial. Для этого в программе терминала нужно выполнить команду:

```
sudo apt-get install python-serial
```

Листинг 4.31. Сценарий Python для передачи сообщений на плату Arduino

```
#!/usr/bin/env python

import serial
from time import sleep

ser = serial.Serial('/dev/serial0', 9600)
ser.write('P13=1')
sleep(1)
ser.write('P13=0')
```

После запуска сценария на компьютере Raspberry Pi встроенный светодиод платы Arduino должен начать мигать с периодом в 1 секунду.

Обсуждение работы решения и возможных проблем

Скетч Arduino определяет начало сообщения по получению символа `P`, а затем с помощью функции `parseInt()` извлекает из полученного сообщения номер контакта и его требуемое состояние. Сообщение `P13=1` включает встроенный светодиод (который подключен к контакту 13), а сообщение `P13=0` выключает его. Дополнительная информация по обмену данными по последовательному интерфейсу и использованию функции `parseInt()` излагается в предыдущем материале этой главы.

Во многих платах Arduino и совместимых платах встроенный светодиод подключен к иному контакту, а не к контакту 13. Поэтому, чтобы не заморачиваться с поисками правильного номера контакта, в скетче Arduino для версии формата сообщения `B=0/1` (без указания номера контакта) вместо номера контакта используется константа `LED_BUILTIN`.

Таким образом, если встроенный светодиод вашей платы подключен к иному контакту, чем 13, используйте следующий сценарий Python (листинг 4.32), в котором команда `B` мигает светодиодом, подключенным к номеру контакта `LED_BUILTIN`.

Листинг 4.32. Альтернативный сценарий Python для передачи сообщений на плату Arduino

```
#!/usr/bin/env python

import serial
from time import sleep

ser = serial.Serial('/dev/serial0', 9600)
ser.write('B=1')
sleep(1)
ser.write('B=0')
```

Контакты платы компьютера Raspberry Pi не могут работать с сигналами напряжением 5 В, поэтому для подключения этой платы к плате Arduino с напряжением рабочего питания величиной 5 В следует использовать делитель напряжения, показанный в схеме на рис. 4.9. Если же ваша плата Arduino рассчитана на напряжение рабочего питания величиной 3,3 В, этот делитель обычно можно не использовать, но его наличие не причинит плате никакого вреда. Более подробная информация по делителям напряжения приводится в *разд. 5.11*.

В нашем случае на Arduino отправляются очень простые сообщения, но их можно расширить, чтобы управлять почти всеми функциями платы Arduino и отправлять с нее информацию обратно на плату Raspberry Pi. Дополнительная информация по организации взаимодействия платы Arduino с компьютером по последовательному интерфейсу приводится в *разд. 4.0*.

Более подробную информацию по работе с языком программирования Python и одноплатным компьютером Raspberry Pi можно найти в Интернете и в соответствующей литературе — например, в книге Саймона Монке[7] «Raspberry Pi Cookbook» («Рецептурный справочник по Raspberry Pi», третье издание).

Почему может казаться, что Arduino работает быстрее, чем Raspberry Pi?

Одноплатный компьютер Raspberry Pi — воистину поразительный образец передовых технологий. Он способен работать под управлением развитой операционной системы — например, Linux или Windows 10, что не под силу стандартной плате Arduino. Эта возможность может оказаться весьма важной, если необходимо использовать сложные приложения — например, для распознавания речи, визуального сопоставления образов и бесчисленных других задач, поддерживаемых Linux и Windows. Но когда требуется программное управление входными или выходными состояниями контактов с высокой точностью и скоростью, то в некоторых ситуациях плата Arduino может реагировать быстрее, чем плата Raspberry Pi.

Это объясняется тем, что управление состояниями контактов платы Raspberry Pi осуществляется через множественные слои программного обеспечения, предназначенные для изолирования аппаратного обеспечения от операционной системы. Одним из таких слоев является

[7] Вас также может заинтересовать книга по работе с Raspberry Pi этого же автора: Саймон Монк. Мейкерство. Arduino и Raspberry Pi: пер. с англ. СПб.: Изд-во «БХВ-Петербург» (**https://bhv.ru/product/mejkerstvo-arduino-i-raspberry-pi/**).

используемый язык программирования. И эти накладные расходы замедляют скорость управления состояниями контактов. Кроме того, поскольку операционная система постоянно прерывает исполнение каждой задачи, чтобы выполнить какую-либо другую задачу, обращения к контактам для управления задачей могут испытывать небольшие, но непостоянные задержки.

Для базовой платы Arduino Uno можно добиться постоянной частоты переключения состояний контактов величиной 8 МГц (см. *разд. 18.11*). Когда Джунас Пихладжама (Joonas Pihlajamaa) тестировал с помощью набора эталонных тестов (**https://oreil.ly/QghxL**) частоту переключения состояний контактов ввода/вывода общего назначения платы Raspberry Pi, он, используя Python и библиотеку RPi.GPIO, смог получить всего лишь частоту 70 КГц. Однако применение собственных библиотек Raspberry Pi и языка программирования C позволяет получить результаты даже лучшие, чем для платы Arduino Uno. Используя этот подход, Джунас смог добиться частоты 22 МГц.

Более современные платы Arduino и совместимые платы поддерживают еще более высокую скорость переключений. Например, плата Teensy 3 может переключать состояния контактов с частотой 48 МГц.

ГЛАВА 5

Простой ввод цифровых и аналоговых данных

5.0. Введение

Способность платы Arduino воспринимать входные цифровые и аналоговые сигналы позволяет ей реагировать на действия пользователя и на события окружающей среды. В этой главе представлены методы, которые можно использовать для мониторинга таких входных сигналов и реагирования на них: фиксация нажатия кнопки, считывание входных сигналов с цифровой клавиатуры и определение диапазона значений напряжения.

Мы также познакомимся с контактами платы Arduino, которые могут обнаруживать входные цифровые и аналоговые сигналы. Цифровые контакты ввода позволяют обнаруживать наличие или отсутствие на них напряжения определенного уровня, а аналоговые контакты ввода позволяют измерять присутствующее на них напряжение в определенном диапазоне значений.

На рис. 5.1 показана схема расположения контактов платы Arduino Uno. Такая схема расположения контактов также используется многими совместимыми с Arduino платами, включая линию плат Metro компании Adafruit и платы компании SparkFun. На странице продуктов Arduino (**https://oreil.ly/aZPSA**) приводится список официальных плат со ссылками на веб-страницы с технической информацией на каждую из них, включая схемы расположения их контактов. Если ваша плата отсутствует в этом списке, информацию по схеме расположения ее контактов следует искать на веб-сайте ее поставщика или производителя.

Определение наличия на цифровом контакте высокого (`HIGH`) или низкого (`LOW`) уровня напряжения осуществляется с помощью функции `digitalRead()` языка Arduino. Для плат с напряжением питания +5 В (например, Arduino Uno) высоким логическим уровнем считается напряжение от 3 до 5 В, а для плат с напряжением питания 3,3 В (например, плат с процессором на ARM-архитектуре) — напряжение от 2 до 3,3 В. Низким логическим уровнем для всех плат является напряжение величиной 0 В. Конфигурирование контактов Arduino для работы в *режиме ввода* осуществляется с помощью функции `pinMode()`, которой в параметрах передается номер требуемого контакта и установленный режим — например: `pinMode(`*`номер_контакта`*`, INPUT)`.

Платы с расположением контактов в стиле Arduino Uno (включая плату Arduino Leonardo, многие платы Metro компании Adafruit и плату RedBoard компании SparkFun) оснащены 14 цифровыми контактами ввода/вывода общего назначения.

Эти контакты пронумерованы от 0 до 13, как показано в верхней части платы на рис. 5.1. На платах Uno и совместимых на 100% платах (это обычно платы с микроконтроллером ATmega328) контакты 0 и 1 (обозначены RX и TX) используются для интерфейса USB-Serial, и их не следует использовать для других целей. Дополнительная информация по последовательному интерфейсу приведена в *главе 4*.

Язык Arduino для обозначения многих контактов содержит константы в виде их логических названий. В табл. 5.1 приводится список этих констант, которые можно использовать во всех случаях, когда требуется указать номер соответствующего контакта. Скорее всего, в коде примеров, с которым вам придется столкнуться, будут использоваться номера контактов. Но учитывая громадное разнообразие плат

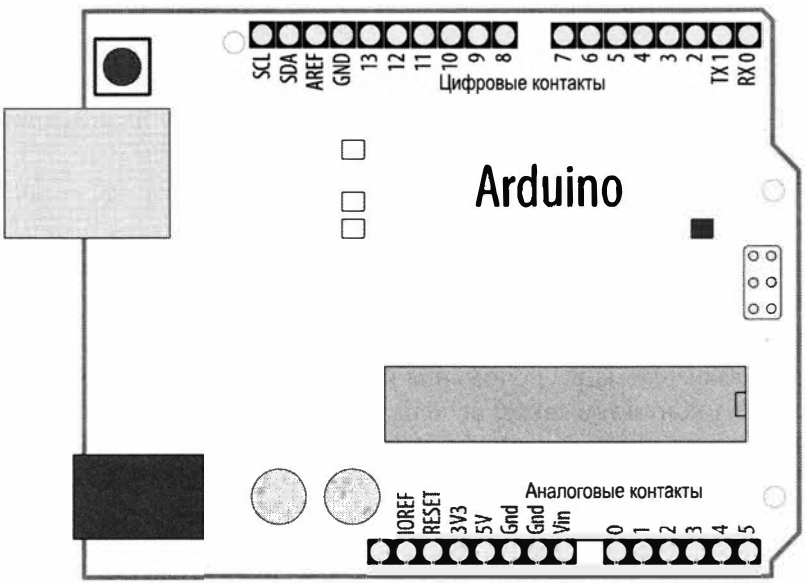


Рис. 5.1. Схема расположения цифровых и аналоговых контактов общего ввода/вывода платы Arduino Uno

Таблица 5.1. *Константы для контактов с определенной функциональностью*

Константа	Функциональность	Константа	Функциональность
A0	Аналоговый ввод 0	LED_BUILTIN	Встроенный светодиод
A1	Аналоговый ввод 1	SDA	Данные I^2C
A2	Аналоговый ввод	SCL	Сигнал тактирования I^2C
A3	Аналоговый ввод	SS	Выбор ведомого устройства интерфейса SPI
A4	Аналоговый ввод	MOSI	Ввод интерфейса SPI
A5	Аналоговый ввод	MISO	Вывод интерфейса SPI
		SCK	Сигнал тактирования интерфейса SPI

Arduino и совместимых плат, следует избегать использования в коде цифровых обозначений контактов и применять вместо них соответствующие константы. Например, на плате Arduino Uno для аналогового ввода 0 используется контакт 14, но на плате MKR WiFi 1010 для этой цели служит контакт 15, а на плате Arduino Mega — контакт 54. Однако ко всем этим контактам можно обращаться посредством соответствующей константы — A0.

Аналоговые контакты можно использовать в качестве цифровых. При этом обращение к ним осуществляется по их символическим константам. Например: `pinMode(A0, INPUT);`.

Некоторые платы (например, Mega и Due) оснащены бóльшим количеством цифровых и аналоговых контактов, чем стандартная базовая плата Arduino Uno. Но на этих платах расположение цифровых контактов с 0 по 13 и аналоговых контактов с 0 по 5 такое же, как и на стандартной плате, чтобы на них можно было устанавливать шилды, разработанные для стандартных плат. Как и на стандартной плате, аналоговые контакты этих плат можно использовать в качестве цифровых, но на плате Mega эти контакты пронумерованы от A0 по A15. Схема расположения цифровых и аналоговых контактов платы Mega показана на рис. 5.2.

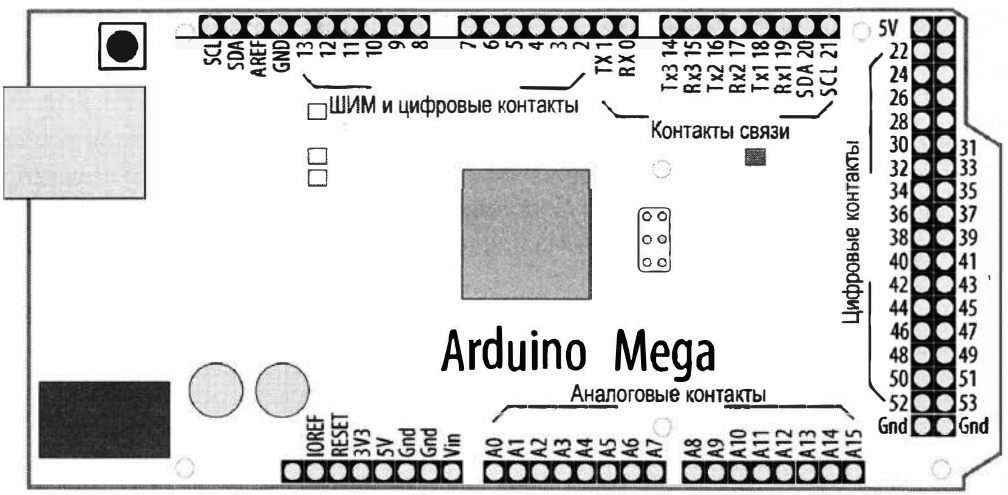


Рис. 5.2. Схема расположения цифровых и аналоговых контактов общего ввода/вывода платы Arduino Mega

Платы Arduino Uno, Leonardo и многие другие платы оснащены встроенным светодиодом, подключенным к цифровому контакту 13, но на других платах этот светодиод может быть подключен к другому контакту. Поэтому, чтобы обеспечить совместимость скетчей с разными платами, при обращении к этому контакту для управления встроенным светодиодом следует всегда использовать его константу `LED_BUILTIN`, а не собственно номер контакта. Если ваша плата не оснащена встроенным светодиодом, о том, как подключить внешний светодиод к цифровому контакту, рассказано в *разд. 7.1*. При этом для управления внешним светодиодом также

нужно будет изменить идентификатор его контакта управления с `LED_BUILTIN` на номер используемого контакта.

В примерах с использованием входных цифровых сигналов иногда задействуются внутренние или внешние подтягивающие резисторы для установки на контакте известного состояния (уровня), когда на него не подается активный сигнал. Без такого резистора контакт будет находиться в *плавающем* состоянии, т. е. непостоянном. В результате функция `digitalRead()` может сначала возвратить высокий уровень, а буквально миллисекунды спустя — низкий, независимо от того, задействован ли этот контакт или нет (например, нажатием кнопки). Подтягивающие резисторы бывают двух типов: повышающие и понижающие:

- *повышающий* резистор подключается одним выводом к шине положительного питания, в результате чего на контакте устанавливается постоянный высокий уровень (5 или 3,3 В — в зависимости он величины напряжения питания конкретной платы). Кнопка управления таким контактом подключается другим своим выводом к шине «земли». В результате при нажатии кнопки на контакте устанавливается низкий уровень. При отпущенной кнопке на контакте удерживается высокий уровень, подаваемый на него через повышающий резистор.
- *понижающий* резистор подключается одним выводом к шине «земли», вследствие чего на контакте устанавливается постоянный низкий уровень величиной 0 В. Кнопка управления таким контактом подключается другим своим выводом к шине питания. В результате при нажатии кнопки на контакте устанавливается высокий уровень.

В качестве подтягивающих резисторов обычно используются резисторы номиналом 10 кОм, но любой номинал в диапазоне от 4,7 до 20 кОм или более прекрасно справится с поставленной задачей. Дополнительная информация об используемых в этой главе компонентах представлена в *приложении 1*.

Работа с электронными компонентами

Это первая из многих глав, в которой рассматривается подключение электронных компонентов к плате Arduino. Если у вас нет опыта работы с электроникой, рекомендуем ознакомиться с материалами, приведенными в *приложении 1* — по электронным компонентам, в *приложении 2* — по принципиальным схемам и документации (datasheets), в *приложении 3* — по сборке схем и в *приложении 5* — по диагностированию неполадок с аппаратными компонентами. Информация по этим темам также приводится во многих учебниках для начинающих. Ближе всего нам в этом плане две из таких книг: «Getting Started with Arduino» («Знакомство с Arduino») авторы Massimo Banzi (Массимо Банци) и Michael Shiloh (Майкл Шило), изданной сообществом Make (Make Community)[1], и книга «Making Things Talk», автор Tom Igoe (Том Иго), издательство Make Community[2]. Начальные знания по темам электроники, рассматриваемым в этой и следующих главах, также можно получить из книг: «Getting Started in

[1] К сожалению, эта книга не издана на русском языке, поэтому мы рекомендуем вам в качестве адекватной замены книгу Джереми Блума «Изучаем Arduino: инструменты и методы технического волшебства». 2-е изд.: пер. с англ.: **https://bhv.ru/product/izuchaem-arduino-instrumenty-i-metody-tehnicheskogo-volshebstva-2-e-izd-per-s-angl/**.

[2] Доступен русский перевод этой книги: Иго Том. Arduino, датчики и сети для связи устройств (2-е изд. Изд-во «БХВ-Петербург»): **https://bhv.ru/product/arduino-datchiki-i-seti-dlya-svyazi-ustrojstv-2-e-izd/**.

Electronics» («Электроника для начинающих»), автор Forrest M. Mims III (Форрест Мимз), издательство Master Publishing, «Make: Electronics» , автор Charles Platt (Чарльз Платт), издательство Make Community[3] и «Physical Computing» («Физические вычисления»), автор Tom Igoe (Том Иго), издательство Cengage.

Если вам никогда раньше не приходилось собирать электронные схемы в общем и подключать компоненты к плате Arduino в частности, вам нужно быть особенно внимательным при подключении аппаратных устройств к плате Arduino, в частности при подключении линий питания. Микроконтроллер платы Arduino — достаточно стойкая микросхема, которая может выдержать определенный объем неправильного обращения, но ее можно вывести из строя, подав на нее напряжение неправильной полярности или закоротив какой-либо выходной контакт. Платы Arduino с 32-разрядной архитектурой обычно немного менее стойкие. Микроконтроллеры на таких платах, как Uno, используют питание напряжением 5 В, поэтому на контакты этих плат нельзя подавать сигналы с напряжением, превышающим напряжение питания. Но большинство более новых плат Arduino и совместимых с ними плат могут выдерживать самое большее 3,3 В. Чтобы узнать максимальное напряжение, которое можно подавать на контакты ввода/вывода вашей платы, обратитесь к документации на эту плату.

Микроконтроллер некоторых плат Arduino установлен в гнездо, что позволяет в случае повреждения извлечь его и заменить новым, не выбрасывая всю плату. Если вы новичок в электронике и хотите экспериментировать, разумно использовать такие платы. Но в платах Arduino Uno Rev3 SMD[4] микроконтроллер впаян в плату, что делает его замену невозможной.

Микроконтроллеры плат Arduino оснащены встроенными повышающими резисторами, которые можно активировать, передавая функции `pinMode()` параметр `INPUT_PULLUP`, как показано в скетче из листинга 5.3. При этом нет надобности использовать внешние повышающие резисторы.

В отличие от цифровых значений (сигналов), которые имеют только два состояния: включенное и выключенное, аналоговые сигналы непрерывно изменяются и могут принимать бесконечное множество значений в границах определенного диапазона значений. Хорошим примером аналогового сигнала являет уровень громкости динамика — он не просто включен или выключен, но может иметь любое значение громкости между этими пределами. Многие датчики предоставляют информацию об измеряемом ими параметре окружающей среды, варьируя свое выходное напряжение в соответствии с полученными ими замерами физических или иных таких параметров. Выходной сигнал датчика подается на вход аналогового контакта платы Arduino, где он считывается функцией `analogRead()`, которая возвращает значение, величина которого пропорциональна присутствующему на контакте входному напряжению. Это значение будет находиться в диапазоне от 0 до 1023, что соответствует пределам напряжения входного сигнала 0 и 5 В (или 3,3 В — для плат с напряжением питания 3,3 В). Величина значения, возвращаемого функцией `analogRead()` для напряжений между этими предельными напряжениями, прямо пропорциональна поступившему напряжению. Например, для напряжения 2,5 В (половина от максимального предела 5 В) функция возвратит значение приблизитель-

[3] Доступен русский перевод этой книги: Платт Ч. Электроника для начинающих. 2-е изд. СПб.: Изд-во «БХВ-Петербург» (**https://bhv.ru/product/elektronika-dlya-nachinayushhih-2-e-izd/**), а также перевод ее продолжения (Make: More Electronics) — Платт Ч. Электроника. Логические микросхемы, усилители и датчики для начинающих. СПб.: Изд-во «БХВ-Петербург» (**https://bhv.ru/product/elektronika-logicheskie-mikroshemy-usiliteli-i-datchiki-dlya-nachinayushhih/**).

[4] SMD (от *англ.* Surface Mounted Device) — устройство для поверхностного монтажа.

но 511 (половина верхнего предела 1023). Плата Arduino оснащена шестью аналоговыми вводами (обозначенными от A0 до A5), расположенными в нижней части платы (см. рис. 5.1). При необходимости эти контакты можно также использовать и как цифровые контакты ввода/вывода. В некоторых примерах, демонстрирующих работу с аналоговыми сигналами, для подачи на контакт изменяющегося аналогового сигнала используется *потенциометр* (переменный резистор). Для этой цели лучше всего подойдет потенциометр номиналом 10 кОм.

Хотя большинство схем, представленных в этой главе, сравнительно легко подключить к плате Arduino, рекомендуется обзавестись беспаечной макетной платой, чтобы упростить подключение к плате внешних компонентов. Полноразмерная беспаечная макетная плата имеет 830 гнездовых контактов, в которые вставляются выводы компонентов, и по два ряда гнезд с каждой стороны для шин питания. Для реализации проектов в этой книге прекрасно подойдут беспаечные макетные платы компаний Jameco, Adafruit Industries, Digi-Key и SparkFun, артикулы 20723, 239, 438-1045-ND и PRT-12615 соответственно. Беспаечные макетные платы половинного размера также пользуются популярностью частично благодаря тому, что они приблизительно такого же размера, что и плата Arduino Uno.

Другим полезным инструментом для реализации проектов является недорогой мультиметр. Подойдет практически любая модель, способная измерять напряжение и сопротивление. Но неплохо также иметь функции прозвонки (проверки на проводимость) цепи и измерения тока. Эти возможности предоставляются мультиметрами компаний Jameco, Adafruit Industries, Digi-Key и SparkFun, артикулы 220759, 2034, 1742-1135-ND и TOL-12966 соответственно.

5.1. Работа с кнопочными переключателями

ЗАДАЧА

Требуется, чтобы скетч реагировал на замыкание электрической цепи. Например, при нажатии кнопки, или другого переключателя, или внешнего устройства, создающего электрический контакт.

РЕШЕНИЕ

Определить состояние переключателя, подключенного к цифровому контакту платы Arduino, работающему в режиме ввода, можно с помощью функции `digitalRead()`. Скетч, код которого приводится в листинге 5.1, включает встроенный светодиод при нажатии кнопки. Схема подключения кнопки показана на рис. 5.3.

Листинг 5.1. Включение встроенного светодиода нажатием кнопки

```
/*
Скетч Pushbutton
Нажатие кнопки, подключенной к цифровому контакту 2 платы,
включает встроенный светодиод
*/
```

```
const int inputPin = 2; // Контакт для подключения кнопки

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT); // Задаем выходной режим работы
                                 // для контакта светодиода
   pinMode(inputPin, INPUT);     // Задаем входной режим работы для контакта кнопки
}

void loop()
{
   int val = digitalRead(inputPin); // Считываем входное значение
   if (val == HIGH) // Проверяем наличие высокого входного уровня
   {
      digitalWrite(LED_BUILTIN, HIGH); // Если кнопка нажата, включаем светодиод
   }
   else
   {
      digitalWrite(LED_BUILTIN, LOW); // Если кнопка не нажата, выключаем светодиод
   }
}
```

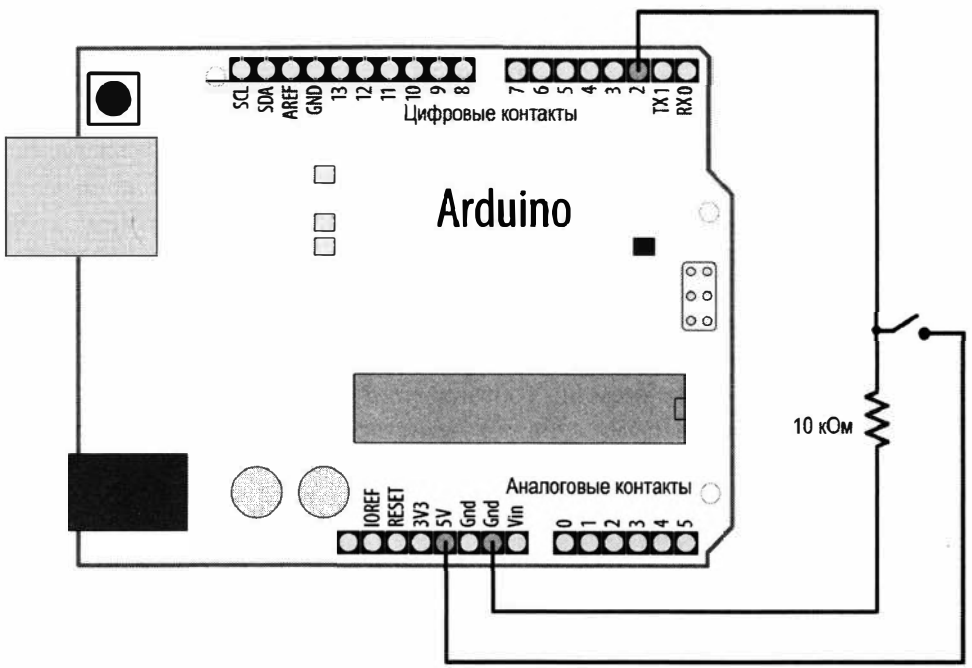


Рис. 5.3. Подключение кнопки с использованием понижающего резистора

Платы Arduino обычно оснащены встроенным светодиодом, подключенным к цифровому контакту ввода/вывода, который идентифицируется константой `LED_BUILTIN`. Но достаточно часто можно встретить код, в котором обращение к контакту встроенного светодиода осуществляется обращением к контакту номер 13. Это пра-

вильный номер контакта для встроенного светодиода платы Uno и многих других плат, но в то же самое время существует много других плат, использующих другой номер контакта. Поэтому, чтобы обеспечить универсальность кода, для обращения к контакту встроенного светодиода следует использовать константу, а не его номер. Информация по подключению внешнего светодиода для плат без встроенного светодиода приводится в *разд. 7.1*. Для управления внешним светодиодом в таком случае нужно изменить идентификатор его контакта управления с `LED_BUILTIN` на номер используемого контакта.

Обсуждение работы решения и возможных проблем

В скетче примера функция `setup()` конфигурирует контакт управления светодиодом для работы в режиме вывода, а контакт кнопки — для работы в режиме ввода.

Чтобы функция `digitalWrite()` могла управлять выходным логическим уровнем контакта, для этого контакта необходимо задать работу в режиме вывода (`OUTPUT`). А чтобы с контакта можно было считывать присутствующий на нем входной логический уровень, для него нужно задать работу в режиме ввода (`INPUT`).

Функция `digitalRead()` считывает логический уровень, присутствующий на контакте ввода (`inputPin`), и возвращает значение `HIGH` при наличии высокого уровня (5 В для большинства 8-разрядных плат и 3,3 В для большинства 32-разрядных плат) и значение `LOW` при наличии низкого уровня (0 В). Высоким (`HIGH`) уровнем считается любой уровень с напряжением от 3 до 5 В (для плат с питанием 5 В) или от 2 до 3,3 В (для плат с питанием 3,3 В). Уровень с меньшим напряжением считается низким (`LOW`). Если контакт, сконфигурированный для работы в режиме ввода, не подключен к какому-либо источнику напряжения (такое состояние контакта называется *плавающим*), то его состояние будет неопределенным, и функция `digitalRead()` для такого контакта будет возвращать неопределенное значение, которое может быть или `HIGH`, или `LOW`. Подключив контакт к шине «земли» через понижающий резистор, как показано на рис. 5.3, мы обеспечиваем на нем постоянный сигнал низкого уровня, поскольку резистор «подтягивает» напряжение на контакте к напряжению на контакте «земли» (обозначенном меткой GND на большинстве плат), которое равно 0 В. Иными словами, резистор *понижает* напряжение на контакте до 0 В, что и объясняет его название — понижающий. А при нажатии кнопки на контакт напрямую подается напряжение +5 В, в результате чего функция `digitalRead()` возвратит значение `HIGH`.

Не подавайте на цифровые и аналоговые контакты сигналы напряжением выше 5 В (или 3,3 В для плат с напряжением питания 3,3 В). Если вы не знаете напряжение питания своей платы, обратитесь к документации на нее поставщика или производителя. Максимальное допустимое напряжение также можно узнать и на странице платы в онлайновом каталоге поставщика. Более высокое напряжение может повредить схему контакта и даже, возможно, вывести из строя сам микроконтроллер. Кроме этого, будьте внимательны, чтобы не подключить кнопку таким образом, что при ее нажатии напряжение 5 В соединялось бы напрямую (без промежуточного резистора) с «землей». Хотя это может и не повредить микроконтроллер, но короткое замыкание вредно для источника питания.

В скетче из листинга 5.1 полученное функцией `digitalRead()` значение сохраняется в переменной `val`. Это значение будет равно `HIGH` при нажатой кнопке и `LOW` в противном случае.

Используемая в рассматриваемом примере (и почти во всех других примерах этой книги) кнопка замыкает электрическую цепь при нажатии и размыкает, когда отпущена. Такие переключатели называются *нормально разомкнутыми*. Переключатели, замыкающие цепь в отпущенном состоянии и размыкающие ее при нажатии, называются *нормально замкнутыми*.

Когда значение переменной `val` равно `HIGH`, на контакт управления светодиодом подается высокий уровень, включающий его, а когда `LOW` — подается низкий уровень, выключающий светодиод.

Хотя все цифровые контакты ввода/вывода общего назначения по умолчанию сконфигурированы для работы в режиме ввода, лучше всего задавать режим работы контактов в скетче явно — в качестве напоминания самому себе, каким образом используется тот или иной контакт.

Вам может повстречаться подобный код, в котором вместо значения `HIGH` используется значение `true`. Эти значения взаимозаменяемы, и оба также могут иногда выражаться значением 1. Соответственно значение `false` эквивалентно значениям `LOW` и 0. Используйте тот вид значения в своем коде, которое лучше всего выражает логику разрабатываемого приложения.

В схеме примера можно использовать практически любой переключатель, но особой популярностью в проектах Arduino пользуются тактильные кнопки без фиксации положения — благодаря своей низкой цене и возможности вставлять их напрямую в беспаечную макетную плату.

Логику примера из листинга 5.1 можно реализовать и другим способом, показанным в следующем фрагменте кода:

```
void loop()
{
   // Включаем светодиод, если на контакте кнопки высокий уровень,
   // выключаем в противном случае
   digitalWrite(LED_BUILTIN, digitalRead(inputPin));
}
```

В этой версии кода состояние кнопки не сохраняется в переменной, а напрямую управляет состоянием светодиода. Это удобное упрощение, но оно не дает никакого улучшения эффективности кода. Поэтому, если оно вам кажется слишком кратким, используйте тот вариант, который вам более удобен.

Вместо подключения в понижающей конфигурации (см. рис. 5.3) резистор на контакте ввода сигнала с кнопки можно подключить и в повышающей конфигурации, как показано на рис. 5.4. Код для этого варианта подключения кнопки похож на предыдущий, но с обратной логикой — при нажатии кнопки на контакт подается низкий уровень. В листинге 5.2 приводится соответствующий фрагмент кода для работы с такой логикой, модифицированный код в нем выделен полужирным шрифтом.

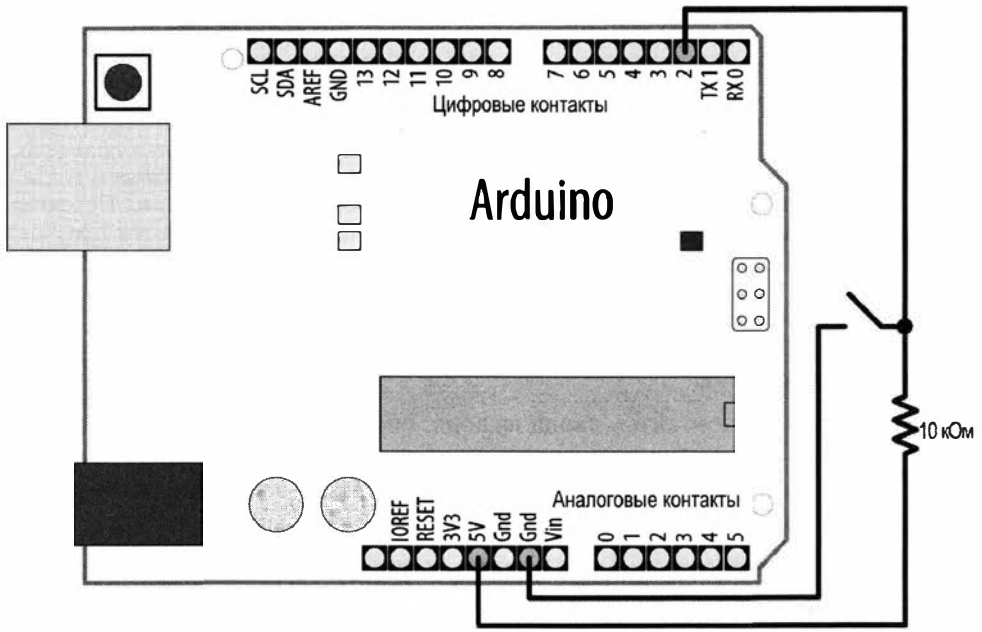


Рис. 5.4. Подключение кнопки с использованием повышающего резистора

Листинг 5.2. Код для работы с низким активным уровнем при нажатии кнопки

```
void loop()
{
   int val = digitalRead(inputPin); // Считываем входное значение
   if (val == HIGH) // Проверяем наличие высокого входного уровня
   {
      digitalWrite(LED_BUILTIN, LOW); // Если кнопка нажата, выключаем светодиод
   }
   else
   {
      digitalWrite(LED_BUILTIN, HIGH); // Включаем светодиод
   }
}
```

Дополнительная информация

Дополнительная информация по рассмотренным в этом разделе функциям и константам приводится на соответствующих страницах справки веб-сайта Arduino:

- функция `digitalRead()` (**https://oreil.ly/Bk6HD**);
- функция `digitalWrite()` (**https://oreil.ly/kWoWV**);
- функция `pinMode()` (**https://oreil.ly/ffTZC**);
- константы (`HIGH`, `LOW` и т. п.) (**https://oreil.ly/QLS77**).

Дополнительная информация по цифровым контактам ввода/вывода общего назначения приводится на соответствующей веб-странице справки веб-сайта Arduino (**https://oreil.ly/XDBur**).

5.2. Подключение кнопки без внешних подтягивающих резисторов

ЗАДАЧА

Вас интересует, возможно ли упростить схему подключения кнопки, удалив из нее внешний подтягивающий резистор.

РЕШЕНИЕ

Как объясняется в *разд. 5.1*, на цифровом вводе необходимо поддерживать постоянный низкий или высокий уровень, чтобы обеспечить на нем постоянное состояние при отпущенной кнопке. Для обеспечения такого постоянного уровня контакт подключается через подтягивающий резистор к шине положительного питания или к шине «земли». Но постоянный высокий уровень на контакте можно обеспечить и без использования внешнего резистора. Микроконтроллер платы Arduino оснащен встроенными повышающими резисторами, которые можно подключать к его контактам, работающим в режиме ввода. Задействовать эти резисторы с требуемым контактом можно посредством функции `pinMode()`, передавая ей в параметрах номер требуемого контакта и значение `INPUT_PULLUP`:

```
pinMode(номер_контакта, INPUT_PULLUP);
```

Пример скетча с задействованием внутреннего (встроенного) повышающего резистора приводится в листинге 5.3. Кнопка для работы с этим скетчем подключается, как показано на рис. 5.5. Это практически такое же подключение, что и на рис. 5.4, но только без внешнего резистора.

Кнопка подключается напрямую к контакту 2 и контакту «земли» (обозначаемому меткой GND на большинстве плат). По определению напряжение на линии «земли» равно 0 В.

Листинг 5.3. Подключение кнопки с использованием внутреннего повышающего резистора

```
/*
Скетч Input pullup
Нажатие кнопки, подключенной к цифровому контакту 2 платы,
включает встроенный светодиод
*/

const int inputPin = 2; // Контакт для подключения кнопки
```

```
void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(inputPin, INPUT_PULLUP); // Подключаем внутренний повышающий
                                    // резистор для контакта кнопки inputPin
}

void loop()
{
   int val = digitalRead(inputPin); // Считываем входное значение
   if (val == HIGH) // Проверяем наличие высокого входного уровня
                    // на контакте
   {
      digitalWrite(LED_BUILTIN, LOW); // Если кнопка не нажата,
                                      // выключаем светодиод
   }
   else
   {
      digitalWrite(LED_BUILTIN, HIGH); // Включаем светодиод
   }
}
```

Плата Arduino имеет несколько контактов «земли» (GND). Все они подключены вместе, поэтому используйте тот, который наиболее удобен.

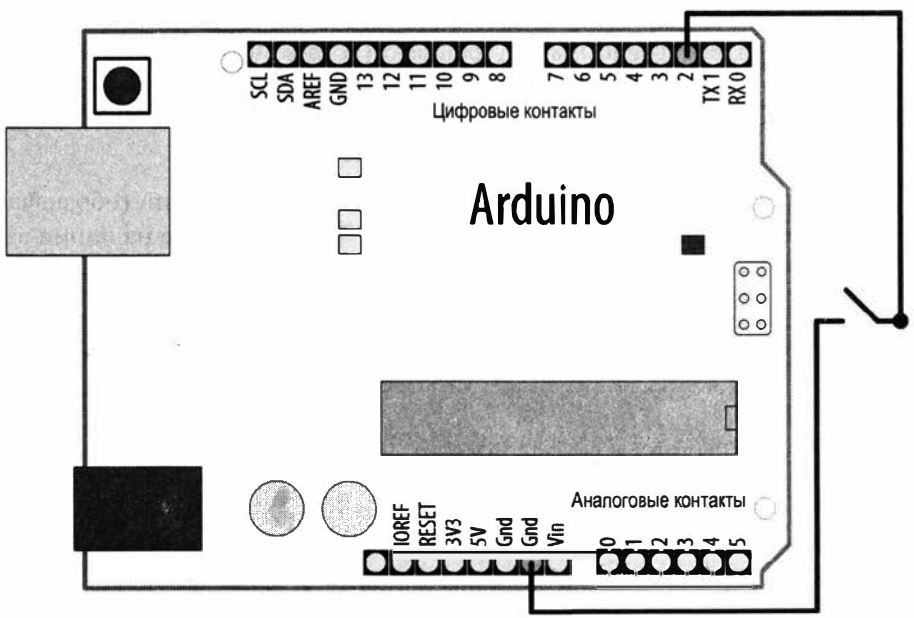


Рис. 5.5. Подключение кнопки с использованием встроенного повышающего резистора

Обсуждение работы решения и возможных проблем

Использование совместно с кнопкой повышающего резистора инвертирует логику управления светодиодом: функция `digitalRead()` будет возвращать значение `LOW` при нажатой кнопке и `HIGH` — при отпущенной. Сопротивление внутренних повышающих резисторов составляет 20 кОм или более (от 20 до 50 кОм). Это сопротивление подходит для большинства приложений, но для некоторых устройств могут требоваться резисторы меньшего сопротивления. Узнать, подходят ли внутренние повышающие резисторы для подключения к плате Arduino определенного внешнего устройства можно, сверившись со документацией (datasheet) на это устройство.

Имейте в виду, что если на платах AVR (таких как Uno) приложение переключает режим работы контакта с ввода на вывод, то на этом контакте остается высокий (`HIGH`) или низкий (`LOW`) уровень. Иными словами, если контакт, работающий в режиме вывода и с установленным на нем высоким уровнем, переключить в режим ввода, то будет включен его повышающий резистор и на контакте установится постоянный высокий уровень. А если на контакте, работающем в режиме вывода, установлен низкий уровень, то при переключении его в режим ввода повышающий резистор не будет задействован. Если для контакта включен повышающий резистор, то переход в режим вывода установит на контакте высокий уровень, что может непреднамеренно включить, например, подключенный к нему светодиод.

5.3. Надежное определение нажатия кнопки (устранение дребезга контактов)

ЗАДАЧА

Требуется надежно определять состояние кнопки, избегая ложных срабатываний, вызываемых *дребезгом* контактов. Дребезг контактов — это явление, когда контакты переключателя вибрируют некоторое время, замыкаясь и размыкаясь, прежде чем установиться в одном стабильном состоянии. Процесс устранения ложных считываний состояния контактов называется *устранением дребезга контактов*.

РЕШЕНИЕ

Существует много способов решения этой проблемы. Один из таких способов приводится в листинге 5.4 для кнопки, подключенной по схеме, показанной на рис. 5.3.

Листинг 5.4. Скетч для устранения дребезга контактов

```
/*
* Скетч Debounce
* Нажатие кнопки, подключенной к цифровому контакту 2 платы,
   включает встроенный светодиод
* Логика устранения дребезга контактов кнопки предотвращает ложное
   чтение состояния кнопки
*/
```

```
const int inputPin = 2; // Номер контакта кнопки
const int debounceDelay = 10; // Количество итераций, в течение которых
                      // можно ожидать установки стабильного уровня на контакте кнопки
bool last_button_state = LOW; // Последнее состояние кнопки
int ledState = LOW; // Замкнутая или разомкнутая (HIGH или LOW)

// Функция debounce возвращает стабильное состояние кнопки
bool debounce(int pin)
{
   bool state;
   bool previousState;

   previousState = digitalRead(pin); // Сохраняем состояние кнопки
   for(int counter=0; counter < debounceDelay; counter++)
   {
      delay(1); // Выжидаем в течение 1 мс
      state = digitalRead(pin); // Считываем состояние контакта
      if( state != previousState)
      {
         counter = 0; // Обнуляем счетчик, если состояние изменилось
         previousState = state; // и сохраняем текущее состояние
      }
   }
   // Переходим сюда, когда состояние контактов стабильно в течение
   // большего времени, чем период устранения дребезга
   return state;
}

void setup()
{
   pinMode(inputPin, INPUT);
   pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
   bool button_state = debounce(inputPin);

   // Если изменилось состояние кнопки, и кнопка была нажата
   if (last_button_state != button_state && button_state == HIGH)
   {
      // Toggle the LED
      ledState = !ledState;
      digitalWrite(LED_BUILTIN, ledState);
   }
   last_button_state = button_state;
}
```

Когда нужно выполнить надежную проверку на нажатие кнопки, следует вызвать функцию debounce(), передав ей в параметре номер контакта, к которому подключе-

на кнопка, для которой выполняется устранение дребезга. Функция возвращает значение HIGH, если кнопка нажата и находится в стабильном состоянии. Если кнопка не нажата или еще не вошла в стабильное состояние, функция возвращает значение LOW.

Обсуждение работы решения и возможных проблем

Функция debounce() проверяет наличие состояния кнопки после задержки, которая должна быть достаточно длительной, чтобы контакты кнопки прекратили вибрировать и приняли стабильное состояние. Для кнопок с большим периодом дребезга может потребоваться большее количество итераций цикла (для некоторый кнопок этот период может занимать целых 50 мс, а то и больше). Функция debounce() проверяет состояние кнопки столько раз, сколько указано в переменной debounceDelay. Если в течение этого периода времени состояние кнопки стабильное, функция возвращает значение HIGH, если кнопка нажата, и LOW, если нет. В булевом контексте, каковым является оператор if, значение HIGH равнозначно значению true, а значение LOW — значению false. Если в течение периода устранения дребезга состояние кнопки меняется, то счетчик обнуляется и проверка начинается сначала. Этот процесс повторяется до тех пор, пока состояние кнопки не перестанет изменяться в течение периода устранения дребезга.

Хотя переменной debounceDelay присвоено значение 10, и в каждой итерации осуществляется задержка длительностью 1 мс, настоящее время задержки всего цикла может превышать 10 мс по двум причинам. Первая (в зависимости от скорости конкретной платы) — для выполнения всех других операций цикла требуется значительное время, которое добавляется к задержке. Вторая — если в процессе исполнения кода цикла состояние кнопки меняется, счетчик цикла обнуляется.

В функции loop() скетча выполняется постоянная проверка состояния кнопки. Если состояние меняется (с HIGH на LOW или наоборот) и если кнопка нажата (находится в состоянии HIGH), скетч переключает состояние светодиода. При этом одно нажатие кнопки включает светодиод, а следующее — выключает его.

Если кнопка подключена с повышающим резистором вместо понижающего (см. рис. 5.5), возвращаемое функцией debounce() значение необходимо инвертировать, поскольку в этой конфигурации состояние нажатой кнопки будет LOW, но для нажатой кнопки функция должна возвращать значение true (равнозначное значению HIGH). В листинге 5.5 приводится код функции debounce() для работы с кнопкой с повышающим резистором. Это практически та же самая функция, что и в листинге 5.4, в которой изменены только последние четыре строки кода (выделены полужирным шрифтом).

Листинг 5.5. Функция устранения дребезга контактов кнопки с повышающим резистором

```
bool debounce(int pin)
{
   bool state;
   bool previousState;
```

```
   previousState = digitalRead(pin); // Сохраняем состояние кнопки
   for(int counter=0; counter < debounceDelay; counter++)
   {
      delay(1); // wait for 1 ms
      state = digitalRead(pin); // Считываем состояние контакта
      if( state != previousState)
      {
         counter = 0; // Обнуляем счетчик, если состояние изменилось
         previousState = state; // и сохраняем текущее состояние
      }
   }
   // Переходим сюда, когда состояние контактов стабильно в течение
   // большего времени, чем период устранения дребезга
   if(state == LOW) // Значение LOW означает, что кнопка нажата
                    // (поскольку используется повышающий резистор)
      return true;
   else
      return false;
}
```

Для тестирования правильной работы функции в скетч можно добавить переменную для отслеживания количества нажатий кнопки. Значение этой переменной можно отображать в окне монитора порта (см. *главу 4*), наблюдая, увеличивается ли оно при каждом нажатии кнопки. Если отображаемое значение нажатий кнопки не совпадает с действительным их количеством, увеличивайте значение переменной debounceDelay до тех пор, пока они не начнут совпадать. В листинге 5.6 приводится код для отображения количества нажатий кнопки для использования с рассмотренной ранее функцией debounce().

Листинг 5.6. Код для отображения в окне монитора порта количества нажатий кнопки

```
int count; // добавьте эту переменную для хранения количества
           // нажатий кнопки

void setup()
{
   pinMode(inPin, INPUT);
   pinMode(LED_BUILTIN, OUTPUT);
   Serial.begin(9600); // добавьте эту строку в функцию setup()
}

void loop()
{
   bool button_state = debounce(inputPin);
   if (button_state)
```

```
    {
      count++;                // Инкрементируем счет
      Serial.println(count); // Отображаем счет в окне монитора порта
    }
    // Если изменилось состояние кнопки и кнопка была нажата
    if (last_button_state != button_state && button_state == HIGH)
    {
      // Переключаем состояние светодиода
      ledState = !ledState;
      digitalWrite(LED_BUILTIN, ledState);
    }
    last_button_state = button_state;
}
```

Функцию `debounce()` можно использовать с любым количеством кнопок, но при этом необходимо, чтобы контакты, к которым подключены кнопки, были в режиме ввода.

Возможный недостаток этого метода для некоторых приложений может заключаться в том, что с момента вызова функции до установления стабильного состояния контактов кнопки невозможно исполнение какого бы то ни было другого кода. В большинстве случаев это не играет роли, но иногда скетчу может требоваться обслуживать другие процессы, пока ожидается стабилизация контактов кнопки. Решение этой проблемы приводится в *разд. 5.4*.

Дополнительная информация

Среда Arduino IDE содержит пример скетча для устранения дребезга контактов. Чтобы открыть его, выполните команду меню **Примеры | 02.Digital | Debounce**.

5.4. Определение длительности нажатия кнопки

ЗАДАЧА

Приложению требуется определить, сколько времени кнопка находится в текущем состоянии. Например, вам надо увеличивать некое значение, пока кнопка удерживается нажатой, при этом чем дольше кнопка остается нажатой, тем быстрее выполняется это увеличение. Такой подход применяется во многих электронных часах для установки значений параметров времени. Или же нужно знать, достаточно ли долго была нажата кнопка, чтобы установились стабильные показатели (см. *разд. 5.3*).

РЕШЕНИЕ

Эта задача решается с помощью скетча вычитающего таймера, код которого приводится в листинге 5.7. Кнопка подключается, как показано на рис. 5.5 в *разд. 5.2*.

Листинг 5.7. Скетч вычитающего таймера

```
/*
Скетч SwitchTime
Вычитающий таймер с единицей счета величиной 1/10 секунды
Включает светодиод, когда счет становится 0
Нажатие кнопки инкрементирует счет, а удерживание кнопки
 повышает скорость инкрементации
*/

const int ledPin = LED_BUILTIN; // Номер контакта встроенного светодиода (вывод)
const int inPin = 2;            // Номер контакта кнопки (ввод)
const int debounceTime = 20; // Время в миллисекундах, требуемое для
             // установления стабильного состояния контактов кнопки
const int fastIncrement = 1000; // Повышаем скорость инкрементации
                                // после заданного количества миллисекунд
const int veryFastIncrement = 4000; // Повышаем скорость инкрементации
                   // еще больше после заданного количества миллисекунд
int count = 0; // Декрементация счета происходит каждые 1/10 секунды,
               // пока счет не достигнет 0

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

void loop()
{
   int duration = switchTime();
   if(duration > veryFastIncrement)
   {
      count = count + 10;
   }
   else if (duration > fastIncrement)
   {
      count = count + 4;
   }
   else if (duration > debounceTime)
   {
      count = count + 1;
   }
   else
   {
      // Кнопка не нажата, поэтому обслуживаем счетчик
      if( count == 0)
```

```
      {
         digitalWrite(ledPin, HIGH); // Включаем светодиод, если счет равен 0
      }
      else
      {
         digitalWrite(ledPin, LOW); // Выключаем светодиод, если счет не равен 0,
         count = count - 1;         // и декрементируем счет
      }
   }
   Serial.println(count);
   delay(100);
}

// Возвращаем время (в миллисекундах)
// удержания кнопки нажатой (значение LOW)
long switchTime()
{
   // Это статические переменные. Они рассматриваются более подробно
   // в разд. «Обсуждение работы решения и возможных ошибок»
   static unsigned long startTime = 0; // Когда было обнаружено
                                       // изменение состояния кнопки
   static bool state;                  // Текущее состояние кнопки

   if(digitalRead(inPin) != state)     // Проверяем, изменилось ли состояние кнопки
   {
      state = ! state;      // Да, изменилось, инвертируем состояние
      startTime = millis(); // Сохраняем время
   }
   if(state == LOW)
   {
      return millis() - startTime; // Кнопка нажата, возвращаем
                                   // время в миллисекундах
   }
   else
   {
      return 0; // Возвращаем 0, если кнопка не нажата (на контакте
                // высокий уровень)
   }
}
```

Нажатие кнопки устанавливает таймер, инкрементируя счет таймера, отпускание кнопки запускает обратный отсчет. Код выполняет процедуру устранения дребезга контактов и повышает скорость инкрементации счетчика, когда кнопка удерживается нажатой в течение определенного периода времени. Счет таймера инкрементируется на единицу при начальном нажатии кнопки (после устранения дребезга). Удерживание кнопки нажатой больше одной секунды повышает скорость инкрементации счета в четыре раза, а больше четырех секунд — в 10 раз. Отпускание

кнопки запускает обратный счет, и когда его значение доходит до 0 на контакте устанавливается высокий уровень (что в нашем случае включает светодиод).

Обсуждение работы решения и возможных проблем

Основной частью рассматриваемого решения является функция `switchTime()`, которая возвращает время (в миллисекундах), в течение которого кнопка удерживается нажатой. Поскольку в этом решении кнопка подключена с использованием внутреннего повышающего резистора (см. *разд. 5.2*), при нажатии кнопки на ее контакте установится низкий уровень, который и возвратит функция `digitalRead()`.

Далее в функции `loop()` выполняется проверка значения, возвращенного функцией `switchTime()`, и, в зависимости от этого значения, принимается решение, что делать дальше. Если кнопка удерживалась нажатой в течение времени, требующего самой высокой скорости инкрементации счета (значение `veryFastIncrement`), счетчик инкрементируется с этой скоростью (`count + 10`). В противном случае выполняется проверка по значению `fastIncrement`, и в случае успеха счетчик инкрементируется с соответствующей скоростью (`count +4`). Если же и это условие не выполняется, то производится проверка на удержание кнопки в течение времени, достаточного для прекращения дребезга, и при положительном результате проверки счетчик инкрементируется с самой низкой скоростью (`count +1`). Из всех этих условий может выполняться только одно. Но если не выполняется ни одно условие, это означает, что или кнопка не нажата, или же была нажата в течение слишком короткого времени, чтобы прекратился дребезг контактов. В таком случае выполняется проверка на нуль значения счетчика, и при положительном результате включается светодиод. Если же значение счетчика не равно нулю, значение декрементируется и светодиод выключается.

Функцию `switchTime()` можно использовать только для устранения дребезга контактов кнопки. В листинге 5.8 приводится фрагмент кода, реализующий логику устранения дребезга вызовом функции `switchTime()`.

Листинг 5.8. Устранение дребезга контактов с использованием функции `switchTime()`

```
// Время (в миллисекундах), в течение которого контакты должны
// находиться в стабильном состоянии
const int debounceTime = 20;
if( switchTime() > debounceTime)
{
   Serial.print("switch is debounced"); // Дребезг устранен
}
```

Такой подход может быть удобным при необходимости устранять дребезг контактов для нескольких кнопок, поскольку он может проверить, сколько времени кнопка находится в нажатом состоянии, а затем возвратиться к обработке других задач, пока контакты не примут стабильное состояние. Чтобы реализовать такой много-

кнопочный вариант, необходимо сохранить текущее состояние кнопки (нажата или нет) и время последнего изменения ее состояния. Это можно сделать несколькими способами. В следующем примере мы будем использовать отдельную функцию для каждой кнопки. Переменные для кнопок можно создать в виде глобальных в начале скетча (глобальные переменные доступны из любой части скетча), но более удобно, когда переменные для конкретной кнопки находятся в функции для этой кнопки.

Чтобы сохранить значение переменных, определяемых в функции, в скетче используют *статические переменные*. Статические переменные внутри функции предоставляют постоянное хранилище для значений, которые не должны меняться между вызовами функции. Статическая переменная удерживает присвоенное ей значение даже после возврата функции. Последнее значение, присвоенное статической переменной, будет доступно после следующего вызова функции. В этом смысле статические переменные похожи на глобальные переменные (переменные, объявленные вне функции, обычно в начальной части скетча), с которыми нам приходилось работать в других решениях. Но, в отличие от глобальных переменных, объявленные внутри функции статические переменные доступны только внутри этой функции. Польза статических переменных состоит в том, что их значение нельзя случайно изменить кодом вне их функции.

В скетче из листинга 5.9 показано, как добавить отдельные функции для разных кнопок. Кнопки для этого скетча подключаются, как показано на рис. 5.5 в *разд. 5.2*, причем вторая кнопка подключена к контакту 3 и «земле».

Листинг 5.9. Пример использования индивидуальной функции устранения дребезга для каждой кнопки

```
/*
Скетч SwitchTimeMultiple
Выводит в окно монитора порта длительность одновременного
 нажатия обеих кнопок
*/

const int switchAPin = 2; // Контакт для кнопки A
const int switchBPin = 3; // Контакт для кнопки B

void setup()
{
   pinMode(switchAPin, INPUT_PULLUP);
   pinMode(switchBPin, INPUT_PULLUP);
   Serial.begin(9600);
}

void loop()
{
   unsigned long timeA;
   unsigned long timeB;
```

```
   timeA = switchATime();
   timeB = switchBTime();

   if (timeA > 0 || timeB > 0)
   {
      Serial.print("switch A time=");
      Serial.print(timeA);
      Serial.print(", switch B time=");
      Serial.println(timeB);
   }
}

unsigned long switchTime(int pin, bool &state, unsigned long &startTime)
{
   if(digitalRead(pin) != state) // Проверяем, изменилось ли
                                 // состояние кнопки
   {
      state = ! state;      // Да, изменилось, инвертируем состояние
      startTime = millis(); // Сохраняем время
   }
   if(state == LOW)
   {
      return millis() - startTime; // Возвращаем время в миллисекундах
   }
   else
   {
      return 0; // Возвращаем 0, если кнопка не нажата
                // (на контакте высокий уровень)
   }
}

long switchATime()
{
   // Это статические переменные (см. объяснение ранее в тексте)
   // Когда было впервые обнаружено изменение состояния кнопки
   static unsigned long startTime = 0;
   static bool state; // Текущее состояние кнопки
   return switchTime(switchAPin, state, startTime);
}

long switchBTime()
{
   // Это статические переменные (см. объяснение ранее в тексте)
   // Когда было впервые обнаружено изменение состояния кнопки
   static unsigned long startTime = 0;
   static bool state; // the current state of the switch
   return switchTime(switchBPin, state, startTime);
}
```

Вычисление времени в скетче выполняет функция `switchTime()`. Эта функция исследует и обновляет состояние кнопки и длительность такого состояния. Параметры состояния `state` и начального времени `startTime` передаются функции по ссылке (ссылки рассматриваются в *разд. 2.11*). Для сохранения времени нажатия и состояния каждой кнопки используются отдельные функции: `switchATime()` и `switchBTime()`. Поскольку для хранения значений объявлены статические переменные, значения сохраняются, когда функции завершают работу. Хранение переменных в функции исключает возможность использования неправильной переменной. Переменные для номеров контактов, к которым подключены кнопки, объявлены как глобальные, поскольку их значения требуются для конфигурации режима работы этих контактов. Но вследствие их объявления с использованием ключевого слова `const`, по сути это не переменные, а константы, в результате чего компилятор не позволит модифицировать их. Таким образом, непреднамеренное изменение значений этих переменных кодом скетча исключено.

Задача ограничения внешнего воздействия на переменные становится все более важной по мере повышения сложности проектов. Среда Arduino IDE предоставляет более элегантный способ решения этой задачи с помощью *классов*. Подробное описание реализации этого подхода приводится в *разд. 16.4*.

В функции `loop()` код скетча проверяет, как долго кнопка удерживается в нажатом состоянии. Если любая из кнопок удерживается нажатой больше нуля миллисекунд, в окно монитора порта выводится длительность нажатия каждой кнопки. Если удерживать нажатой одну или обе кнопки, в окне монитора порта будут непрерывно выводиться увеличивающиеся значения длительности нажатия. Если отпустить обе кнопки, функция `switchTime()` возвратит значение 0, в результате чего вывод в окно монитора порта прекратится до тех пор, пока снова не будет нажата одна или обе кнопки.

5.5. Считывание сигналов с цифровой клавиатуры

ЗАДАЧА

Требуется подключить к плате Arduino цифровую клавиатуру, а затем считывать нажатия кнопок. Это может быть клавиатура, подобная 12-кнопочной цифровой панели компании Adafruit (артикул 419).

РЕШЕНИЕ

Подключите контакты строк и столбцов клавиатуры к плате Arduino, как показано на монтажной схеме, приведенной на рис. 5.6.

Считывать нажатия кнопок этой цифровой клавиатуры можно с помощью скетча (листинг 5.10), отображающего нажатую кнопку в окне монитора порта.

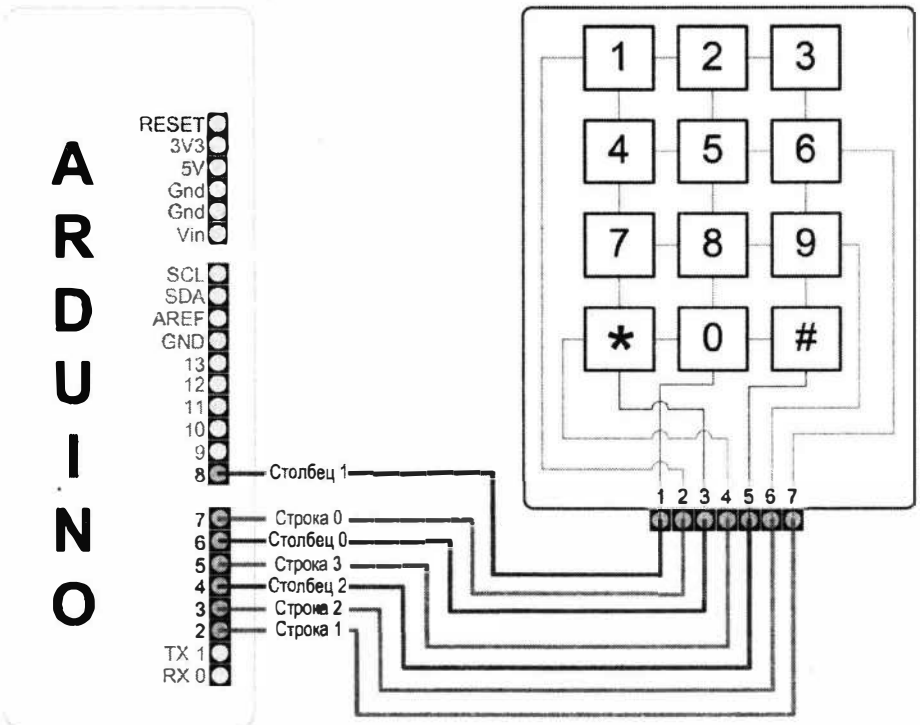


Рис. 5.6. Подключение цифровой клавиатуры к плате Arduino

Листинг 5.10. Считывание нажатия кнопок цифровой клавиатуры

```
/*
Скетч Keypad
Отображает значение нажатой кнопки цифровой клавиатуры
в окне монитора порта
*/

const int numRows = 4; // Количество строк цифровой клавиатуры
const int numCols = 3; // Количество столбцов цифровой клавиатуры
const int debounceTime = 20; // Время в миллисекундах, требуемое
            // для установления стабильного состояния контактов кнопки

// Карта кнопок определяет возвращаемый символ при нажатии
// соответствующей кнопки цифровой клавиатуры
const char keymap[numRows][numCols] =
{
   { '1', '2', '3' } ,
   { '4', '5', '6' } ,
   { '7', '8', '9' } ,
   { '*', '0', '#' }
};
```

```
// Массивы для хранения номеров контактов линий строк и столбцов цифровой клавиатуры
const int rowPins[numRows] = {8, 7, 6, 5}; // Строки с 0-й по 3-ю
const int colPins[numCols] = {4, 3, 2};    // Столбцы с 0-го по 2-й

void setup()
{
   Serial.begin(9600);
   for (int row = 0; row < numRows; row++)
   {
      pinMode(rowPins[row],INPUT_PULLUP); // Задаем режим ввода
                  // для контактов строк и задействуем внутренние повышающие резисторы
   }
   for (int column = 0; column < numCols; column++)
   {
      pinMode(colPins[column],OUTPUT);    // Задаем выходной режим
                                          // работы для контактов столбцов
      digitalWrite(colPins[column],HIGH); // Деактивируем все контакты столбцов
   }
}

void loop()
{
   char key = getKey();
   if( key != 0) { // Если код символа не 0, тогда имеем нажатие кнопки
   Serial.print("Got key "); // Нажата кнопка
   Serial.println(key);
   }
}

// Возвращает значение нажатой кнопки или 0, если нет нажатий
char getKey()
{
   char key = 0; // 0 означает, что не нажата никакая кнопка

   for(int column = 0; column < numCols; column++)
   {
      digitalWrite(colPins[column],LOW);     // Активируем текущий столбец
      for(int row = 0; row < numRows; row++) // Сканируем все строки на нажатие кнопки
      {
         if(digitalRead(rowPins[row]) == LOW) // Кнопка нажата?
         {
            delay(debounceTime); // Устраняем дребезг
            while(digitalRead(rowPins[row]) == LOW);
            // Ждем отпуска кнопки
            key = keymap[row][column]; // Запоминаем нажатую кнопку
         }
      }
```

```
      digitalWrite(colPins[column],HIGH); // Деактивируем текущий столбец
   }
   return key; // Возвращает значение нажатой кнопки или 0,
               // если нет нажатой кнопки
}
```

Этот скетч будет работать должным образом только в том случае, если схема подключения линий строк и столбцов цифровой клавиатуры к контактам платы Arduino (табл. 5.2) соответствует приведенному в нем коду. Если вы используете другую цифровую клавиатуру, уточните схему ее линий строк и столбцов в документации на нее. Будьте внимательны при подключении линий клавиатуры к плате, поскольку неправильное подключение может закоротить контакты и вывести из строя микроконтроллер платы.

Таблица 5.2. *Подключение линий строк и столбцов цифровой панели к контактам платы Arduino*

Контакт платы Arduino	Контакт цифровой панели	Строка/столбец цифровой панели
2	7	Столбец 2
3	6	Столбец 1
4	5	Столбец 0
5	4	Строка 3
6	3	Строка 2
7	2	Строка 1
8	1	Строка 0

Обсуждение работы решения и возможных проблем

Цифровые клавиатуры обычно содержат нормально разомкнутые кнопки, которые при нажатии замыкают строку со столбцом. (Нормально разомкнутая кнопка замыкает электрическую цепь только при ее нажатии.) На рис. 5.6 ранее показано, как внутренние проводники клавиатуры подключают строки и столбцы кнопок к ее разъему. Каждая из четырех строк клавиатуры подключена к контакту платы, работающему в режиме ввода, а каждый столбец подключен к контакту, работающему в режиме вывода. Входной режим работы контактов строк задается в функции `setup()`, при этом одновременно включаются внутренние повышающие резисторы этих контактов (внутренние повышающие резисторы рассматриваются в *разд. 5.0* этой главы).

Функция `getkey()` последовательно устанавливает низкий уровень (`LOW`) на каждом контакте столбцов и для каждого из них последовательно проверяет контакты строк на наличие на них низкого уровня. Поскольку для контактов строк задействованы повышающие резисторы, на этих контактах обычно будет присутствовать высокий уровень (`HIGH`), если только не нажата какая-либо кнопка. (Замыкание кон-

тактов кнопки устанавливает низкий уровень на соответствующем контакте ввода платы Arduino.) Наличие низкого уровня на контакте строки означает замыкание кнопки для столбца и строки. Для устранения дребезга контактов кнопки выдерживается пауза (см. *разд. 5.3*), затем код ожидает освобождения кнопки, после чего в массиве `keymap` определяется нажатая кнопка, которая и возвращается функцией. Если не нажата никакая кнопка, возвращается значение 0.

Библиотека Keypad для Arduino (**https://oreil.ly/FVuL_**) упрощает работу с цифровыми клавиатурами, имеющими другое количество кнопок, и ее можно применять для использования некоторых контактов совместно с символьным жидкокристаллическим дисплеем. Для использования библиотеки ее необходимо установить с помощью Менеджера библиотек среды Arduino IDE (подробная информация по установке библиотек приводится в *разд. 16.2*).

Дополнительная информация

Дополнительная информация по 12-кнопочной цифровой панели компании Adafruit приводится на веб-странице этого устройства (**https://oreil.ly/yD8Lh**).

5.6. Считывание аналоговых сигналов

ЗАДАЧА

Требуется считать входное напряжение, присутствующее на аналоговом контакте. Например, чтобы получить показания потенциометра или другого датчика с выходным сигналом в виде изменяющегося напряжения.

РЕШЕНИЕ

Эта задача решается скетчем, код которого показан в листинге 5.11. С помощью функции `analogRead()` скетч считывает напряжение на аналоговом контакте A0 и мигает светодиодом с частотой, пропорциональной величине возвращенного функцией значения. Напряжение на контакте изменяется с помощью потенциометра, подключенного по схеме, показанной на рис. 5.7.

Листинг 5.11. Считывание значения входного аналогового сигнала

```
/*
Скетч Pot
Мигает светодиодом с частотой, заданной вращением потенциометра
*/

const int potPin = A0;          // Контакт для подключения потенциометра
const int ledPin = LED_BUILTIN; // Контакт для подключения светодиода
int val = 0; // Переменная для хранения значения, полученного
             // с датчика (потенциометра)
```

```
void setup()
{
   pinMode(ledPin, OUTPUT); // Задает выходной режим работы контакта,
                            // к которому подключен встроенный светодиод
}

void loop()
{
   val = analogRead(potPin);   // Считываем напряжение на контакте потенциометра
   digitalWrite(ledPin, HIGH); // Включаем светодиод
   delay(val);   // Удерживаем светодиод включенным в течение времени
                 // (в миллисекундах), заданного сигналом с потенциометра
   digitalWrite(ledPin, LOW); // Выключаем светодиод
   delay(val);    // Удерживаем светодиод выключенным в течение
                  // такого же времени, что держали его включенным
}
```

Если ваша плата рассчитана на питание 3,3 В, а не 5 В, не подключайте потенциометр к контакту 5V, даже если такой контакт имеется на плате. Многие платы с питанием 3,3 В оснащены контактом выходного питания напряжением 5 В, который подключен непосредственно к линии питания, подаваемого на USB-разъем. Этот контакт можно использовать для подачи питания на устройства, требующие рабочее питание напряжением 5 В, но нужно быть внимательным и никогда не подключать контакт выходного питания напряжением 5 В к контактам, которые рассчитаны на 3,3 В. Для плат с напряжением питания 3,3 В потенциометр нужно подключить к контакту выходного питания, обозначенному 3V3.

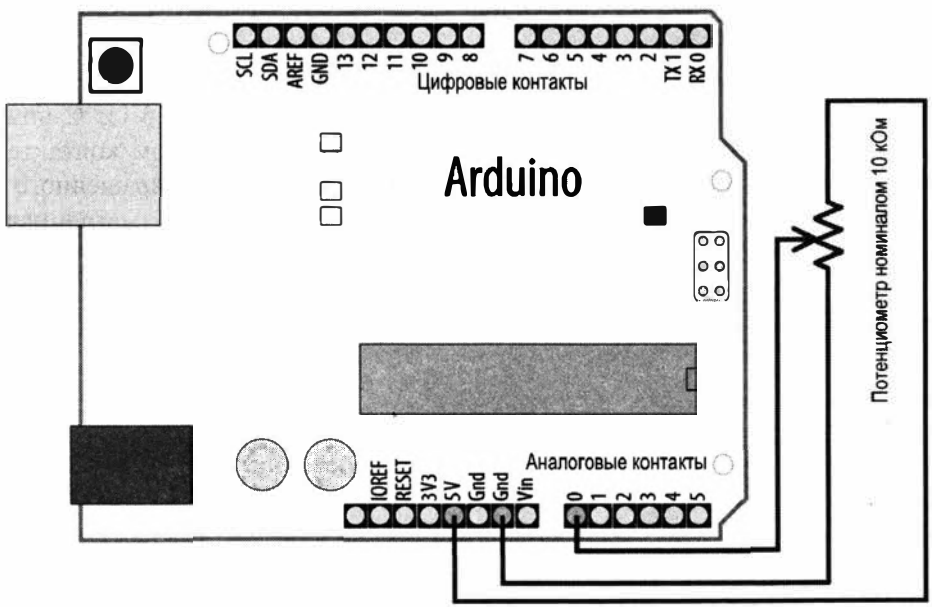


Рис. 5.7. Подключение потенциометра к плате Arduino

Обсуждение работы решения и возможных проблем

В скетче используется функция `analogRead()` для считывания напряжения с контакта ползунка (среднего контакта) потенциометра. Потенциометр имеет три вывода, два из которых подключены к противоположным концам резистивного материала, а третий (обычно средний) подключен к ползунку, который посредством вращения вала потенциометра можно установить в любой точке между крайними точками резистивного материала. При перемещении ползунка от одного края резистивного материала к другому сопротивление между выводом ползунка и одним из крайних выводов увеличивается, а между выводом ползунка и другим крайним выводом уменьшается. Принципиальная схема на рис. 5.7 может помочь понять описанный принцип работы потенциометра. При перемещении ползунка (линия со стрелкой посередине символа резистора) к нижнему краю резистора сопротивление между ползунком и нижним выводом будет уменьшаться, а между ползунком и верхним выводом — увеличиваться. В результате напряжение, подаваемое через потенциометр на аналоговый контакт, будет уменьшаться вплоть до напряжения «земли» 0 В. А перемещение ползунка вверх будет иметь противоположный эффект, и напряжение на контакте будет повышаться вплоть до 5 В (или 3,3 В для плат с питанием 3,3 В).

Если при вращении вала потенциометра по часовой стрелке напряжение на контакте понижается, а не повышается, необходимо поменять местами подключение крайних выводов потенциометра (к контактам 5V и GND).

Функция `analogRead()` измеряет величину напряжения на контакте и возвращает числовое значение, прямо пропорциональное напряжению на контакте. Это значение будет 0 при напряжении на контакте, равном 0 В, и 1023 — при напряжении 5 В (или 3,3 В для плат с напряжением питания 3,3 В, как в случае с большинством 32-разрядных плат). Величина значения, возвращаемого функцией `analogRead()` для напряжений между этими предельными величинами, будет прямо пропорциональна отношению величины напряжения на контакте к максимальному напряжению 5 В (или 3,3 В, в зависимости от используемой платы).

В нашем случае для подачи сигнала на аналоговый контакт лучше всего подойдет потенциометр номиналом 10 кОм.

Контакт, к которому подключается потенциометр, не обязательно конфигурировать для работы в режиме ввода. Эта конфигурация выполняется автоматически при каждом вызове функции `analogRead()`.

Дополнительная информация

В *приложении 2* приводятся некоторые советы по чтению принципиальных схем.

Дополнительная информация по функции Arduino `analogRead()`приводится на соответствующей веб-странице справки Arduino (**https://oreil.ly/TNayy**).

Важная информация, связанная с материалом этого раздела, также предлагается в книге «Getting Started with Arduino» («Знакомство с Arduino») авторов Massimo Banzi (Массимо Банци) и Michael Shiloh (Майкл Шило), изданной сообществом Make (Make Community)[5].

5.7. Масштабирование значения к другому диапазону

ЗАДАЧА

Требуется масштабировать значение, полученное, например, функцией `analogRead()` с потенциометра или с другого устройства, выдающего изменяющееся напряжение. Например, вы хотите отобразить положение вала потенциометра в виде процентного отношения в диапазоне от 0 до 100%.

РЕШЕНИЕ

Функция `map()` языка Arduino позволяется масштабировать значения к требуемому диапазону. Скетч из листинге 5.12 демонстрирует использование этой функции. Скетч считывает выходное напряжение потенциометра в переменную `val` и масштабирует его к диапазону значений от 0 до 100. Скетч также мигает встроенным светодиодом с частотой, пропорциональной напряжению потенциометра, и отображает отмасштабированное значение в окне монитора порта (об отправке данных на монитор порта рассказано в *разд. 4.2*). Потенциометр подключается к плате Arduino, как показано на рис. 5.7 в *разд. 5.6*.

Листинг 5.12. Масштабирование получаемого значения

```
/*
* Скетч Map
* Масштабирует диапазон аналоговых значений сигнала потенциометра
  к диапазону значений от 0 до 100
* Мигает светодиодом пропорционально масштабированному значению
  с периодом от 0 до 100 мс
* Выводит в монитор порта угол поворота вала потенциометра в процентах
*/

const int potPin = A0;     // Контакт для подключения потенциометра
int ledPin = LED_BUILTIN; // Контакт для подключения светодиода

void setup()
{
   pinMode(ledPin, OUTPUT); // Задаем выходной режим работы контакта,
                            // к которому подключен встроенный светодиод
```

[5] К сожалению, эта книга не издана на русском языке, поэтому мы рекомендуем вам в качестве адекватной замены книгу Джереми Блума «Изучаем Arduino: инструменты и методы технического волшебства». 2-е изд.: пер. с англ. (**https://bhv.ru/product/izuchaem-arduino-instrumenty-i-metody-tehnicheskogo-volshebstva-2-e-izd-per-s-angl/**).

```
   Serial.begin(9600);
}

void loop()
{
   int val;      // Переменная для значения, получаемого
                 // с датчика (потенциометра)
   int percent; // Переменная для отмасштабированного значения

   val = analogRead(potPin); // Считываем напряжение на контакте
         // потенциометра (значение val в диапазоне от 0 до 1023)

   percent = map(val,0,1023,0,100); // Значение percent будет
                                    // в диапазоне от 0 до 100
   digitalWrite(ledPin, HIGH);      // Включаем встроенный светодиод
   delay(percent);                  // Удерживаем его включенным
                                 // в течение времени, задаваемым значением percent
   digitalWrite(ledPin, LOW);       // Выключаем светодиод
   delay(100 - percent);            // Удерживаем его выключенным
                    // в течение времени (100 минус время включения)
   Serial.println(percent);         // Выводим в окно монитора порта
                   // угол поворота (в процентах) вала потенциометра
}
```

Обсуждение работы решения и возможных проблем

В *разд. 5.6* рассматривается метод преобразования положения вала потенциометра в значение. А здесь эта тема расширяется, демонстрируя, как использовать функцию `map()` для масштабирования полученного значения к значению в требуемом диапазоне. В частности, в приведенном решении значение, выдаваемое функцией `analogRead()` (в диапазоне от 0 до 1023), масштабируется в значение в диапазоне от 0 до 100. Это масштабированное значение используется для установки *коэффициента заполнения* включения встроенного светодиода. Коэффициент заполнения — это процентное отношение времени, в течение которого светодиод находится во включенном состоянии, ко времени, называемым *периодом* (или циклом), длительность которого в нашем случае составляет 100 мс. Время выключенного состояния светодиода вычисляется вычитанием времени включенного состояния из времени цикла (периода). Таким образом, возвращенное функцией `analogRead()` значение (например, 620) масштабируется функцией `map()` в значение 60. В результате светодиод будет включен в течение 60 мс и выключен в течение 40 мс (100 – 60).

Функция `analogRead()` возвращает значения в диапазоне от 0 до 1023 для напряжений в диапазоне от 0 до 5 В (до 3,3 В для плат с напряжением питания 3,3 В), но для исходного и конечного диапазонов масштабирования можно использовать любые соответствующие предельные значения. Например, вал типичного потенциометра вращается только на 270 градусов от одного предельного состояния до дру-

гого. Отображать угол проворота такого потенциометра можно с помощью следующего кода:

```
int angle = map(val,0,1023,0,270); // Угол поворота потенциометра,
                                   // полученный масштабированием значения,
                                   // возвращенного функцией analogRead()
```

Пределы диапазонов также могут быть и отрицательными значениями. Например, мы хотим отображать 0, когда вал потенциометра находится в среднем положении, отрицательные значения, когда он поворачивается против часовой стрелки, и положительные для поворота по часовой стрелке. Это можно реализовать с помощью следующего кода:

```
// Отображение угла поворота в диапазоне 270 градусов с центром в 0
   angle = map(val,0,1023,-135,135);
```

Функция `map()` может быть полезной в тех случаях, когда требуемый входной диапазон значений не начинается с нуля. Например, когда нужно показать оставшийся заряд батареи пропорционально напряжению батареи в диапазоне от 1,1 до 1,5 В. Это задачу можно реализовать с помощью кода, приведенного в листинге 5.13.

Листинг 5.13. Масштабирование исходного диапазона, не начинающегося с нуля

```
const int board_voltage = 5.0; // Для плат с питанием 3,3 В используйте значение 3.3

const int empty = 1.1/(5.0/1023.0); // Напряжение 1,1 В (1100 мВ),
                                    // когда батарея разряжена
const int full = 1.5/(5.0/1023.0);  // Напряжение 1,5 В (1500 мВ),
                                    // когда батарея полностью заряжена

int val = analogRead(potPin); // Считываем напряжение
int percent = map(val, empty, full, 0,100); // Масштабируем напряжение в проценты
Serial.println(percent);
```

При использовании с функцией `map()` показаний датчика необходимо определить предельные значения, выдаваемые датчиком. Получить минимальное и максимальное значения диапазона датчика можно, отображая их в окне монитора порта. Введите полученные таким образом данные в качестве нижнего и верхнего пределов исходного диапазона в параметры функции `map()`.

Если пределы исходного диапазона нельзя определить заранее, их можно получить посредством калибровки датчика. Один из способов такой калибровки описывается в *разд. 8.11*, а еще один приводится в скетче примера **Calibration** (меню **Примеры | 03. Analog | Calibration**).

Следует иметь в виду, что при передаче функции `map()` значения вне пределов исходного диапазона возвращаемое ею значение также будет вне пределов выходного диапазона. Эта проблема решается с помощью функции `constrain()`, применение которой рассматривается в *разд. 3.5*.

Функция `map()` выполняет целочисленные вычисления, поэтому возвращает только целые числа в пределах заданного диапазона. Любая дробная часть результата отсекается, а не округляется.

Дополнительная информация о том, как возвращаемые функцией `analogRead()` значения соотносятся с собственно напряжением, приведена в *разд. 5.9*.

Дополнительная информация

Более подробная информация по функции `map()` приводится на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/CE9TM**).

5.8. Считывание свыше шести входных аналоговых сигналов

ЗАДАЧА

Требуется считывать значения с большего количества датчиков, чем доступное количество аналоговых контактов. Стандартная плата Arduino оснащена шестью аналоговыми контактами (плата Mega имеет 16 аналоговых контактов), которых может быть недостаточно для конкретного приложения. Например, требуется считывать показания с восьми потенциометров, чтобы управлять восемью параметрами в приложении.

РЕШЕНИЕ

Выбирать требуемый источник аналогового сигнала из нескольких и подключать его к аналоговому контакту платы можно с помощью микросхемы *мультиплексора*. Последовательно выбирая каждый из нескольких источников, можно по очереди считать значения их всех. Скетч для реализации этого решения приводится в листинге 5.14. В качестве мультиплексора применяется популярная микросхема 4051, подключение которой к плате Arduino показано на рис. 5.8. В частности, источники входящих аналоговых сигналов подключаются к контактам микросхемы с Ch0 по Ch7. При этом ни в коем случае нельзя допускать, чтобы напряжение на входных контактах микросхемы превышало 5 В. Неиспользуемые входные контакты микросхемы мультиплексора следует замкнуть на «землю» через резисторы номиналом 10 кОм.

Листинг 5.14. Чтение нескольких источников аналогового сигнала с помощью микросхемы мультиплексора

```
/*
* Скетч multiplexer
* Последовательно считывает 8 аналоговых сигналов в один аналоговый
  контакт платы Arduino, используя микросхему мультиплексора
*/
```

```
// Массив для хранения номеров контактов, используемых для выбора
// одного из восьми вводов мультиплексора
const int select[] = {2,3,4}; // Массив номеров контактов, подключенных
                              // к линиям выбора ввода микросхемы 4051
const int analogPin = A0;     // Номер контакта, подключенного к выводу мультиплексора

// Эта функция возвращает аналоговое значение для заданного канала
int getValue(int channel)
{
   // Устанавливаем (HIGH) и обнуляем (LOW) контакты выбора,
   // чтобы получить двоичное значение канала исходного сигнала
   for(int bit = 0; bit < 3; bit++)
   {
      int pin = select[bit]; // Контакт платы Arduino, подключенный
                             // к контакту выбора канала мультиплексора
      int isBitSet = bitRead(channel, bit); // Значение true, если этот
                                            // бит канала установлен
      digitalWrite(pin, isBitSet);
   }
return analogRead(analogPin);
}

void setup()
{
   for(int bit = 0; bit < 3; bit++)
   {
      pinMode(select[bit], OUTPUT); // Задаем выходной режим работы
               // трех контактов платы для выбора источника сигнала
   }
   Serial.begin(9600);
}

void loop ()
{
   // Отображаем в окне монитора порта считанные значения для каждого
   // источника сигнала (канала)
   for(int channel = 0; channel < 8; channel++)
   {
      int value = getValue(channel);
      Serial.print("Channel ");
      Serial.print(channel);
      Serial.print(" = ");
      Serial.println(value);
   }
   delay (1000);
}
```

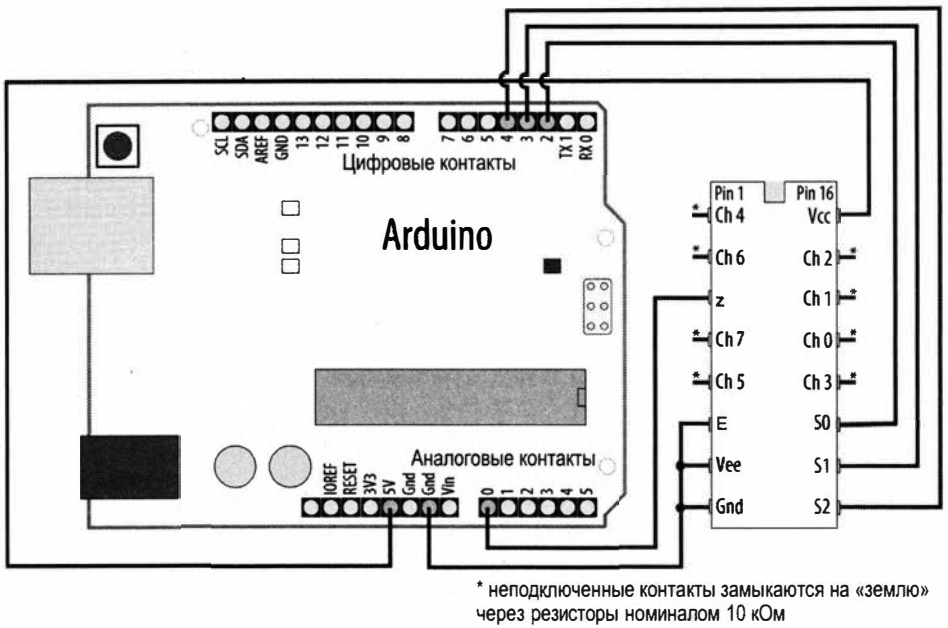


Рис. 5.8. Подключение микросхемы мультиплексора 74HC4051 к плате Arduino

Обсуждение работы решения и возможных проблем

Аналоговый мультиплексор — это переключатель аналоговых сигналов с цифровым управлением. Мультиплексор 4051 выбирает один из восьми входящих сигналов (с Ch0 по Ch7), устанавливая соответствующую комбинацию логических уровней на трех контактах выбора (S0, S1 и S2). С тремя контактами выбора возможны восемь разных комбинаций, которые скетч и устанавливает последовательно, как показано в табл. 5.3.

Таблица 5.3. *Таблица возможных комбинаций логических уровней на контактах выбора мультиплексора 4051*

Контакты выбора			Входной канал
S2	**S1**	**S0**	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Как можно видеть, последовательности в табл. 5.3 являются двоичными представлениями десятичных значений от 0 до 7.

Для получения правильных показаний с датчиков, их линии «земли» необходимо подключить к линиям «земли» мультиплексора 4051 и платы Arduino. При подаче питания на все устройства с контакта выходного напряжения 5V или 3V3 платы Arduino убедитесь в том, что общая потребляемая мощность не превышает максимальную мощность источника питания или максимальную мощность, которую может обеспечить контакт выходного напряжения (в зависимости от того, какая из них меньшая). Например, контакт выходного напряжения 5V платы Arduino Uno может надежно предоставить 900 мА при подаче питания на плату Arduino из внешнего источника питания, или максимум 400 мА — при питании платы через USB-разъем. Но если внешний источник питания может обеспечить всего лишь, например, 500 мА, то максимальный доступный ток для периферийных устройств (мультиплексора и датчиков) будет меньшим, чем 500 мА, поскольку сама плата Arduino (микроконтроллер, светодиоды и прочие компоненты) также потребляет ток. Поэтому общая потребляемая мощность периферийных устройств должна быть соответственно меньшей. Чтобы узнать возможности по току вашей платы и ее отдельных контактов, вам, возможно, нужно будет обратиться к документации на нее, а также к документации на микроконтроллер и стабилизатор напряжения платы. Если окажется, что общее потребление тока приближается к максимальному, который может предоставить плата, используйте отдельные источники питания для подключаемых к ней устройств.

В скетче решения (см. листинг 5.14) функция `getValue()` устанавливает уровни на контактах выбора источника аналогового сигнала с помощью функции:

```
digitalWrite(pin, isBitSet);
```

и считывает аналоговое значение из очередного выбранного источника, используя функцию:

```
analogRead(analogPin);
```

Код для создания битовых последовательностей использует встроенную функцию `bitRead()` (см. *разд. 3.12*).

Следует иметь в виду, что рассмотренный метод выбирает и считывает источники аналоговых сигналов последовательно, поэтому для него требуется больше времени между считываниями источников по сравнению с прямым использованием функции `analogRead()`. При наличии восьми источников сигналов чтение каждого из них займет в восемь раз больше времени. Вследствие этого рассмотренный здесь метод может быть непригодным для считывания значений быстро изменяющихся аналоговых сигналов.

Дополнительная информация

Дополнительная информация по микросхеме мультиплексора 74HC4051 приводится в учебном пособии на веб-странице Arduino Playground (**https://oreil.ly/2wM2x**).

Подробная информация по микросхеме мультиплексора 74HC4051 содержится в ее справочном листке (**https://oreil.ly/ikGzY**).

5.9. Измерение напряжений до 5 В

ЗАДАЧА

Требуется измерять и отображать значение напряжения в диапазоне от 0 до 5 В. Например, показывать в окне монитора порта текущее напряжение батарейки с номинальным напряжением 1,5 В.

РЕШЕНИЕ

Подключите батарейку к контактам платы Arduino, как показано на рис. 5.9. Для считывания входного напряжения, присутствующего на аналоговом контакте, используется функция `analogRead()`. Возвращенное функцией числовое значение преобразовывается в значение напряжения на основе соотношения между считанным значением и эталонным напряжением (5 В).

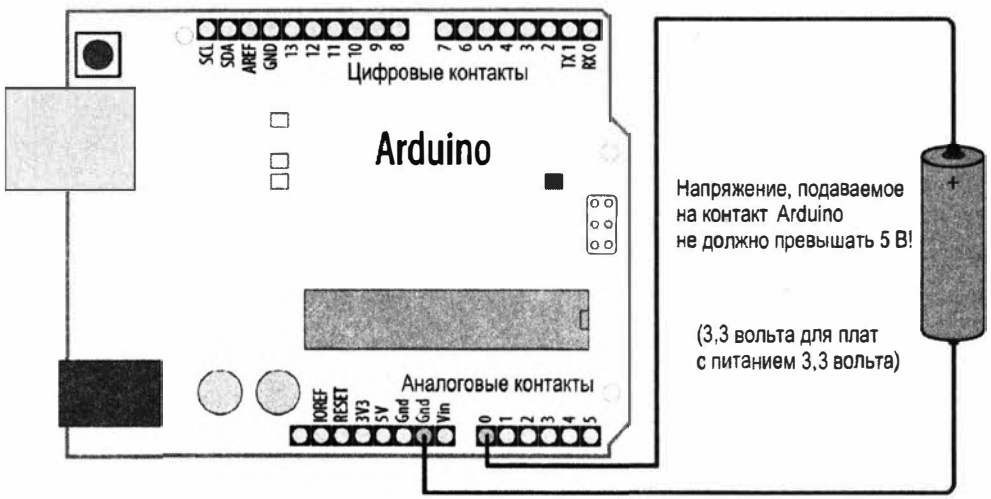


Рис. 5.9. Измерение напряжений до 5 В с помощью платы Arduino с питанием 5 В

Измерение напряжения на плате ESP8266

В случае использования платы ESP8266 диапазон измеряемых напряжений может быть ограничен величинами от 0 до 1 В. При этом некоторые такие платы оснащены встроенным делителем напряжения, позволяющим измерять напряжения до 3,3 В (сама плата ESP8266 работает на напряжении 3,3 В), поэтому обязательно узнайте возможности своей платы на этот счет в ее документации. Максимальное напряжение, которое можно подавать на аналоговые контакты плат ESP8266, не оснащенных делителем напряжения, не должно превышать 1 В (дополнительный материал на тему делителя напряжения излагается в *разд. 5.11*).

В листинге 5.15 приводится скетч простейшего решения, которое вычисляет напряжение, используя операции с плавающей точкой (запятой), а затем отображает результат в окне монитора порта.

Листинг 5.15. Простейшее решение измерения напряжения

```
/*
* Скетч Display5vOrless
* Измеряет напряжение сигнала на аналоговом контакте и отображает
  значение в окне монитора порта
* На контакты Arduino нельзя подавать напряжение свыше 5 В
*/

const float referenceVolts = 5.0; // Эталонное напряжение по умолчанию
                                  // для платы с питанием 5 В
const int batteryPin = A0; // Плюс батареи подключается
                           // к аналоговому контакту 0

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

void loop()
{
   int val = analogRead(batteryPin);    // Считываем значение напряжения
                                        // сигнала на аналоговом контакте
   float volts = (val / 1023.0) * referenceVolts; // Вычисляем напряжение в вольтах
   Serial.println(volts);  // Выводим в монитор порта напряжение в вольтах
}
```

Напряжение в вольтах вычисляется по следующей формуле:

```
вольты = (считанное значение / максимальное значение) x эталонное напряжение
```

При отображении значения с плавающей точкой в окне монитора порта посредством функции `println()` оно форматируется до двух десятичных знаков.

Для плат с питанием 3,3 В внесите в скетч следующее изменение:

```
const float referenceVolts = 3.3;
```

Значения с плавающей точкой занимают большой объем памяти, поэтому, если только числа с плавающей точкой уже не используются где-то в другом месте скетча, будет более эффективным переориентироваться на целочисленные значения. В листинге 5.16 приводится фрагмент кода для вычисления напряжения в милливольтах с использованием целых чисел. На первый взгляд этот код может выглядеть несколько странным, но поскольку функция `analogRead()` возвращает значе-

ние 1023 для напряжения 5 В, то один шаг значения будет равен 5/1023. В милливольтах это будет 5000/1023.

Листинг 5.16. Вычисление напряжения в милливольтах с использованием целых чисел

```
const int batteryPin = A0;

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

void loop()
{
   long val = analogRead(batteryPin); // Считываем значение напряжения
                                      // сигнала на аналоговом контакте
                           // Примечание: переменная val имеет тип long int
   Serial.println( (val * (500000/1023L)) / 100); // Значение в милливольтах
}
```

Для плат с питанием 3,3 В измените `(500000/1023L)` **на** `(330000/1023L)`.

В листинге 5.17 приводится код скетча, который вычисляет напряжение, используя целые числа, и отображает значение напряжения, отформатированное с десятичной точкой. Например, напряжение 1,5 В отображается как 1.5.

Листинг 5.17. Вычисление напряжения с использованием целых чисел и отображение значения с десятичной точкой

```
const int batteryPin = A0;

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

void loop()
{
   int val = analogRead(batteryPin);    // Считываем значение напряжения
                                        // сигнала на контакте

   long mv = (val * (500000/1023L)) / 100; // Вычисляем значение в милливольтах
   Serial.print(mv/1000); // Отображаем целую часть значения напряжения
   Serial.print('.');
```

```
    int fraction = (mv % 1000); // Вычисляем дробную часть
    if (fraction == 0)
    {
       Serial.print("000"); // Добавляем три нуля
    }
    else if (fraction < 10) // Если дробная часть < 10, 0 игнорируется,
             // создавая неправильное значение, поэтому добавляем нули
    {
       Serial.print("00"); // Добавляем два нуля
    }
    else if (fraction < 100)
    {
       Serial.print("0");
    }
    Serial.println(fraction); // Отображаем дробную часть
}
```

Обсуждение работы решения и возможных проблем

Функция `analogRead()` возвращает значение, величина которого прямо пропорциональна отношению измеряемого напряжения к эталонному напряжению (5 В для платы Arduino Uno). Чтобы избежать использования чисел с плавающей точкой (запятой), но при этом сохранить точность измерений, вместо вольт код работает с милливольтами (1 вольт содержит 1000 милливольт). Поскольку значение 1023 означает 5000 милливольт, то каждый шаг диапазона представляет 5000 мВ/1023, т. е. 4,89 милливольта.

Для преобразования в милливольты значений, возвращаемых функцией `analogRead()`, можно применять как число 1023, так и число 1024. Число 1024 обычно используется инженерами, поскольку оно представляет количество возможных значений в диапазоне от 0 до 1023. Но число 1023 более интуитивно для некоторых пользователей, поскольку это наибольшее значение диапазона. На практике же величина вносимых аппаратурой погрешностей больше, чем разница между результатами вычислений с использованием этих двух чисел. Поэтому используйте то из них, с которым вам удобнее работать.

Чтобы избавиться от десятичной запятой, значения умножаются на 100. Иными словами, умножив 5000 милливольт на 100 и разделив полученное произведения на 1023, получим количество милливольт, умноженное на 100. Разделив это значение на 100, получим значение в милливольтах. Если умножение дробных чисел на 100, чтобы позволить компилятору работать с целыми числами, кажется вам слишком запутанным подходом, можете без проблем использовать более медленный и более затратный в аспекте памяти метод с плавающей запятой.

В рассмотренном решении подразумевается использование платы Arduino Uno или подобной 8-разрядной платы с напряжением питания 5 В. Для плат с напряжением питания 3,3 В максимальное измеряемое напряжение составляет 3,3 В без использования делителя напряжения. Подход с использованием делителя напряжения рассматривается в *разд. 5.11*.

5.10. Реагирование на изменения напряжения

ЗАДАЧА

Требуется отслеживать один или несколько источников сигналов и предпринимать некоторое действие, когда напряжение сигнала поднимается выше или опускается ниже определенного порогового значения. Например, при разрядке батареи до определенного уровня требуется начать мигать светодиодом и при этом, возможно, повышать частоту мигания по мере того, как напряжение приближается к критическому уровню.

РЕШЕНИЕ

Для этого решения можно использовать схему, показанную на рис. 5.7 в *разд. 5.9*, но в здесь мы будем сравнивать значение, возвращенное функцией `analogRead()`, с определенным пороговым значением, чтобы определить, не опустилось ли оно ниже этого предела. Скетч решения, код которого приводится в листинге 5.18, начинает мигать светодиодом, когда напряжение понижается до 1,2 В, и ускоряет мигание по мере понижения напряжения. При достижении нижнего порогового значения мигание прекращается и светодиод остается постоянно включенным.

Листинг 5.18. Реагирование на меняющееся напряжение

```
/*
* Скетч RespondingToChanges
* Мигает светодиодом, указывая низкий уровень заряда
*/

long batteryFull = 1500;    // Кол-во милливольт полного заряда батареи
long warningThreshold = 1200;  // Уровень предупреждения
                               // в милливольтах - светодиод мигает
long criticalThreshold = 1000; // Критический уровень
                               // заряда - светодиод горит постоянно

const int batteryPin = A0;
const int ledPin = LED_BUILTIN;

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

void loop()
{
   int val = analogRead(batteryPin); // Считываем значение напряжения
                                     // сигнала на контакте
```

```
   int mv = map(val, 0, 1023, 0, 5000);
   if(mv < criticalThreshold) {
   digitalWrite(ledPin, HIGH);
}
else if (mv < warningThreshold)
{
   int blinkDelay = map(mv, criticalThreshold, batteryFull, 0, 250);
   flash(blinkDelay);
}
else
{
   digitalWrite(ledPin, LOW);
}
delay(1);
}

// Функция для мигания светодиодом с заданным временем включенного
// и выключенного состояния
void flash(int blinkDelay)
{
   digitalWrite(ledPin, HIGH);
   delay(blinkDelay);
   digitalWrite(ledPin, LOW);
   delay(blinkDelay);
}
```

Обсуждение работы решения и возможных проблем

Скетч этого решения масштабирует значение (в диапазоне от 0 до 1023), считанное функцией `analogRead()` с аналогового контакта, к соответствующему значению в диапазоне эталонного напряжения (от 0 В до 5000 милливольт). В частности, при эталонном напряжении 5 В и уровне предупреждения 1 В нам нужно знать, когда возвращенное числовое значение соответствуют одной пятой эталонного напряжения. Функция `map()` возвратит значение 1000 (1000 милливольт = 1 вольт), когда функция `analogRead()` возвратит значение 205.

Когда значение напряжения опустится ниже критического уровня `criticalThreshold`, светодиод переходит в постоянно включенное состояние. Если напряжение выше критического уровня, скетч проверяет, не ниже ли оно уровня предупреждения `warningThreshold`. Если ниже, тогда скетч вычисляет паузу для мигания, масштабируя напряжение (в милливольтах) к соответствующему значению в диапазоне от 0 до 250. Чем ближе измеренное значение к критическому пороговому значению `criticalThreshold`, тем короче пауза мигания, вследствие чего светодиод мигает все быстрее по мере того, как напряжение батареи приближается к критическому уровню. Если же напряжение выше порогового уровня предупреждения `warningThreshold`, светодиод находится в выключенном состоянии.

5.11. Измерение напряжений выше 5 В с помощью делителя напряжения

ЗАДАЧА

Требуется измерить напряжение больше чем 5 вольт. Например, нужно отслеживать уровень заряда батареи номиналом 9 В и подавать сигнал тревоги, включив светодиод, когда напряжение упадет ниже определенного порогового значения.

РЕШЕНИЕ

Решение этой задачи похоже на решение задачи из *разд. 5.9*, но батарея подключается к плате Arduino через делитель напряжения, как показано на рис. 5.10. Для напряжений вплоть до 10 вольт в делителе напряжения можно использовать резисторы номиналом 4,7 кОм. Номиналы резисторов для более высоких напряжений представлены в табл. 5.4. Скетч для работы с делителем напряжения приводится в листинге 5.19.

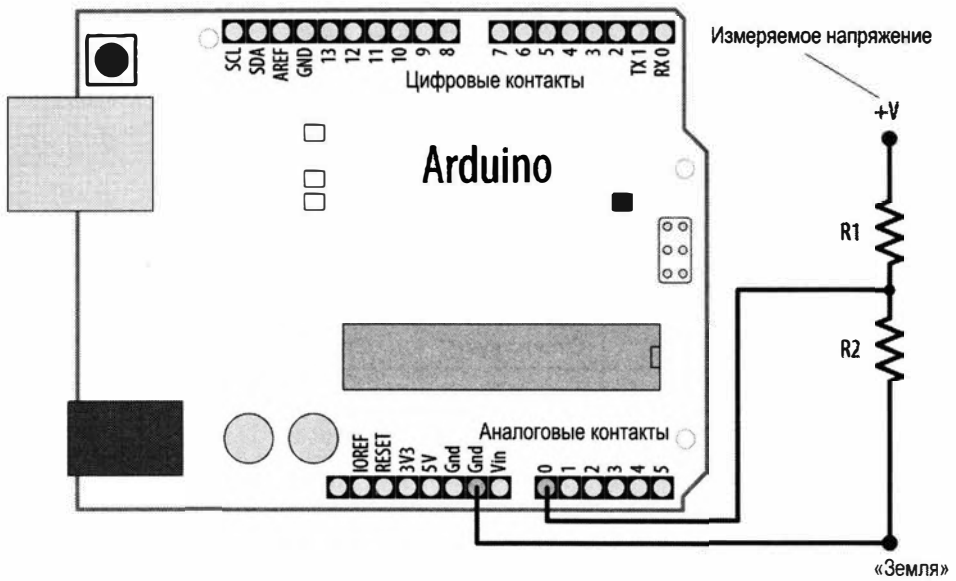


Рис. 5.10. Измерение напряжений, превышающих 5 В, с использованием делителя напряжения

Листинг 5.19. Измерение напряжений, превышающих 5 В, с использованием делителя напряжения

```
/*
Скетч DisplayMoreThan5V
Измеряет напряжение сигнала на аналоговом контакте и отображает
 значение в окне монитора порта
На контакты Arduino нельзя напрямую подавать напряжение выше 5 В
*/
```

```
const float referenceVolts = 5;   // Эталонное напряжение по умолчанию
                                  // для плат с питанием 5 В
//const float referenceVolts = 3.3;  // Эталонное напряжение для плат
                                     // с питанием 3,3 В

const float R1 = 1000; // Значение для максимального напряжения 10 В
const float R2 = 1000;
// Определяем номиналы резисторов делителя (см. текст и табл. 5.4)
const float resistorFactor = 1023.0 * (R2/(R1 + R2));
const int batteryPin = 0; // Плюс батареи подключается к аналоговому контакту 0

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

void loop()
{
   int val = analogRead(batteryPin);  // Считываем значение напряжения
                                      // сигнала на контакте
   float volts = (val / resistorFactor) * referenceVolts; // Вычисляем
                                                          // напряжение в вольтах
   Serial.println(volts);     // Выводим в монитор порта напряжение в вольтах
}
```

***Таблица 5.4.** Номиналы резисторов для напряжений выше 10 В*

Максимальное напряжение	R1	R2	Вычисление R2/(R1 + R2)	Значение `resistorFactor`
5	Перемычка*	Отсутствует**	Отсутствует	1023
10	1 кОм	1 кОм	1(1 + 1)	511
15	2 кОм	1 кОм	1(2 +1)	341
20	3 кОм	1 кОм	1(3 + 1)	255
30	5 кОм (5,1 кОм)	1 кОм	1(5 +1)	170

* +V подключается к аналоговому контакту

** Не подключен

Обсуждение работы решения и возможных проблем

Подобно тому как это делалось в предыдущих примерах измерения напряжения аналоговых сигналов, в этом решении используется то обстоятельство, что возвращаемое функцией `analogRead()` значение является отношением величины измеряемого напряжения к величине эталонного напряжения. Но поскольку измеряемое напряжение разделено двумя понижающими резисторами, чтобы получить действительное значение напряжения, возвращаемое функцией `analogRead()` значение

нужно умножить на некий коэффициент. Код этого скетча похож на код решения в *разд. 5.7*, но считанное с аналогового контакта значение делится не на 1023, а на значение переменной `resistorFactor`:

```
float volts = (val / resistorFactor) * referenceVolts ; // Вычисляем
                                                        // напряжение в вольтах
```

Вычисления для получения номиналов резисторов в табл. 5.4 основаны на следующей формуле: выходное напряжение равно входному напряжению, умноженному на сопротивление резистора R2 и разделенному на сумму сопротивлений резисторов R1 и R2. В примере, в котором используются два резистора одинакового номинала для понижения напряжения с 9 до 4,5 В, величина значения переменной `resistorFactor` составляет 511 (половина 1023). Таким образом, значение переменной `volts` будет вдвое большим, чем входное напряжение на аналоговом контакте. А при использовании резисторов для максимального напряжения 10 В возвращаемое функцией `analogRead()` значение будет приблизительно равно 920.

Подача на контакт платы Arduino напряжения выше 5 В (выше 3,3 В для плат с питанием 3,3 В) может повредить схему контакта и, возможно, вывести из строя микроконтроллер. Поэтому обязательно проверьте, что выбраны правильные значения номиналов резисторов делителя напряжения, прежде чем подавать его выходное напряжение на контакт платы Arduino. А самым лучшим подходом будет всегда измерять напряжение любого сигнала, напряжение которого может превышать 5 В, прежде чем подавать его на контакт платы Arduino.

ГЛАВА 6

Считывание данных с датчиков

6.0. Введение

Считывание и использование данных с датчиков позволяет скетчу Arduino реагировать на события в окружающей среде или докладывать о них. Это одна из наиболее распространенных задач, с которыми вам придется иметь дело. В этой главе предлагаются простые и практические стандартные решения по использованию наиболее популярных устройств ввода и датчиков. В частности, в принципиальных схемах показывается, как подключить устройства и подавать на них питание, а в примерах кода демонстрируется, как использовать получаемые с датчиков данные.

Датчики реагируют на воздействия от физической среды и преобразовывают эти воздействия в электрические сигналы, которые можно подавать на входной аналоговый контакт платы Arduino для считывания. Тип выдаваемого датчиком электрического сигнала зависит от типа датчика и от объема информации, которую он должен передать. Некоторые датчики (например, фоторезисторы и пьезодатчики) выполнены из вещества, которое изменяет свои электрические свойства в ответ на физическое воздействие. Другие датчики представляют собой сложные электронные модули, обрабатывающие получаемый сигнал с помощью встроенного микроконтроллера, прежде чем передать его на Arduino.

Датчики предоставляют информацию плате Arduino посредством следующих способов:

- **Включение и выключение цифрового сигнала**.

  Некоторые устройства — например, такие как датчик наклона из *разд. 6.2* и датчик движения из *разд. 6.4*, просто включают и выключают напряжение. Такие датчики можно рассматривать как простые переключатели, работа с которыми описывается в *главе 5*.

- **Аналоговый сигнал**.

  Другие датчики выдают аналоговый сигнал, напряжение которого пропорционально измеряемой величине, — например, температуре или уровню освещенности. Использование аналоговых датчиков демонстрируется в решениях по определению уровня освещенности (*разд. 6.3*), температуры (*разд. 6.9*) и звука (*разд. 6.8*). Во всех этих решениях задействована функция `analogRead()`, работа с которой рассматривается в *главе 5*.

- **Длительность импульса**.

  Датчики расстояния — например, датчик PING))), рассматриваемый в *разд. 6.5*, предоставляют данные об измеряемом расстоянии до того или иного объекта посредством импульса, длительность которого пропорциональна этому расстоянию. Приложения, работающие с такими датчиками, измеряют длительность выдаваемого ими импульса с помощью функции `pulseIn()`.

- **Последовательная связь.**

  Некоторые датчики предоставляют свою информацию посредством последовательного интерфейса. Например, датчик глобального позиционирования GPS, рассматриваемый в *разд. 6.14*, передает свои данные плате Arduino через последовательный порт (дополнительную информацию по последовательному интерфейсу см. в *главе 4*). Большинство плат Arduino оснащены только одним последовательным портом, и для подключения дополнительных устройств с последовательным интерфейсом необходимо использовать программный последовательный порт. Пример организации такого программного последовательного порта приведен в *разд. 6.14*.

- **Синхронные протоколы I^2C и SPI**.

  Цифровые последовательные интерфейсы связи I^2C и SPI были созданы для того, чтобы позволить процессорам, наподобие тех, которыми оснащена плата Arduino, взаимодействовать с внешними датчиками и модулями. Например, в *разд. 6.15* рассматривается применение протокола I^2C для подключения к плате Arduino модуля гироскопа. Эти протоколы (подробно рассмотренные в *главе 13*) широко используются датчиками, приводами и периферийными устройствами.

Кроме чисто датчиков существует также и другой класс общих измерительных устройств, которые можно использовать в качестве датчиков. Это бытовые устройства, содержащие датчики, но которые предлагаются на рынке в качестве самостоятельных устройств, а не датчиков. Примером такого устройства, рассматриваемого в этой главе, может служить мышь с интерфейсом PS/2. Благодаря тому, что эти устройства обладают датчиками, которые уже встроены в крепкие и эргономические корпуса, они могут быть очень полезными. Они также недорогие (часто даже более дешевые, чем отдельные содержащиеся в них датчики) и производятся в больших количествах. Вполне возможно, что у вас завалялась где-то пара-другая таких устройств.

При использовании устройства, которое непосредственно не рассматривается в решении, проверьте наличие библиотеки для этого устройства в Менеджере библиотек среды Arduino IDE (работа с библиотеками сторонних разработчиков подробно рассматривается в *разд. 16.2*). Даже если библиотека для устройства отсутствует, возможно, решение можно будет приспособить под это устройство, если оно выдает подобный сигнал. Информация о выходном сигнале датчика обычно предоставляется поставщиком или излагается в документации на устройство (которую можно найти в Интернете, выполнив поиск в Google по номеру детали устройства или его описанию).

Документация (справочные листки, datasheets) обычно рассчитана на инженеров, разрабатывающих устройства для промышленного производства, и часто содержит подробности, которые вряд ли необходимы для того, чтобы запустить устройство в работу. Если справочный листок не удастся найти на веб-сайте производителя устройства, его, как правило, можно найти в Интернете, выполнив поиск по названию компонента с добавленным ключевым словом `datasheet`. Информация о характеристиках выходного сигнала датчика обычно содержится в разделе справочного листка, относящемся к формату данных, интерфейсу, выходному сигналу или чему-либо подобному. Не забудьте проверить максимальное напряжение устройства (обычно оно приводится в разделе «Absolute Maximum Ratings», абсолютные предельные значения), чтобы не допустить повреждения устройства или платы.

Датчики с максимальным рабочим напряжением 3,3 В можно повредить, подав на них более высокое напряжение, — например, с контакта платы Arduino с напряжением логического сигнала 5 В. Поэтому проверьте максимальные значения характеристик своего устройства, прежде чем подключать его к плате Arduino. В большинстве случаев сигнал напряжением 5 В можно подключить на вход 3,3 В, используя делитель напряжения. Более подробная информация по работе с делителем напряжения приведена в *разд. 5.11*.

Считывание датчиков, предоставляющих информацию о беспорядочной аналоговой среде, представляет собой смесь науки, искусства и настойчивости. Для получения успешного результата может потребоваться применить находчивость и метод проб и ошибок. Одна из распространенных проблем состоит в том, что датчик просто сообщает о том, что произошло некое физическое событие, но не о том, что вызвало это событие. Способность выбрать правильные условия размещения датчика (место, диапазон, ориентация) и ограничить его подверженность явлениям, воздействия которых на датчик мы не хотим допустить, — это навыки, которые приобретаются с практикой.

Другим вопросом, требующим внимания, является необходимость отделения требуемого сигнала от фонового шума. В *разд. 6.7* показано, как использовать пороговое значение, чтобы определить, когда напряжение сигнала выше определенного уровня, а в *разд. 6.8* — как вычислить среднее значение нескольких считываний, чтобы сгладить шумовые всплески.

Дополнительная информация

Полезная информация по работе с электронными компонентами и сборке схем содержится в книге «Make: Electronics», автор Charles Platt (Чарльз Платт), издательство Make Community[1].

[1] Доступен русский перевод этой книги: Платт Ч. Электроника для начинающих. 2-е изд. СПб.: Изд-во «БХВ-Петербург» (**https://bhv.ru/product/elektronika-dlya-nachinayushhih-2-e-izd/**), а также перевод ее продолжения (Make: More Electronics) — Платт Ч. Электроника. Логические микросхемы, усилители и датчики для начинающих. СПб.: Изд-во «БХВ-Петербург» (**https://bhv.ru/product/elektronika-logicheskie-mikroshemy-usiliteli-i-datchiki-dlya-nachinayushhih/**).

В книге Tom Igoe (Том Иго) «Making Things Talk», издательство Make Community, обсуждается пересечение науки, искусства и настойчивости при разработке и реализации систем с датчиками на основе платформы Arduino[2].

В *разд. 5.0* и *5.6* приводится дополнительная информация по считыванию выходных сигналов с аналоговых датчиков.

6.1. Плата Arduino с несколькими встроенными датчиками

ЗАДАЧА

Плата Arduino Uno, конечно же, обладает многими возможностями, но для получения данных о внешней среде к ней нужно подключать внешние датчики. Было бы здорово, если бы плата Arduino была оснащена несколькими встроенными датчиками.

РЕШЕНИЕ

О вашем желании уже позаботились — представляем плату Arduino Nano 33 BLE Sense. Это недорогая быстродействующая плата малого размера, поддерживающая восемь сенсорных возможностей благодаря группе встроенных в плату компонентов. В табл. 6.1 приводится список этих компонентов, их возможности и название библиотеки поддержки. Прежде чем приступать к работе с платой Arduino Nano 33 BLE Sense, откройте в среде Arduino IDE Менеджер плат и установите пакет Arduino nRF528x Boards (Mbed OS). Инструкции по установке пакетов поддержки

Таблица 6.1. *Встроенные компоненты датчиков платы Arduino Nano 33 BLE Sense*

Компонент	Поддерживаемые возможности	Библиотека
Broadcom APDS-9960	Жесты, определение присутствия, цвет RGB	Arduino_APDS9960
ST HTS221	Температура, относительная влажность	Arduino_HTS221
STLPS22HB	Барометрическое давление	Arduino_LPS22HB
ST LSM9DS1	Блок инерциальных измерений 9D0F (блок IMU[3]): акселерометр, гироскоп, магнетометр	Arduino_LSM9DS1
ST MP34DT05	Цифровой микрофон	Устанавливается по умолчанию с пакетом поддержки платы Nano 33 BLE

[2] Доступен русский перевод этой книги: Иго Т. Arduino, датчики и сети для связи устройств. 2-е изд. СПб.: Изд-во «БХВ-Петербург» (**https://bhv.ru/product/arduino-datchiki-i-seti-dlya-svyazi-ustrojstv-2-e-izd/**).

[3] IMU — от *англ.* Inertial Measurement Unit, инерциальный измерительный блок.

дополнительных плат приводятся в *разд. 1.7*. Затем установите все библиотеки, упомянутые в столбце **Библиотека** табл. 6.1. Инструкции по установке библиотек для дополнительных плат приводятся в *разд. 16.2*.

После установки файлов поддержки для платы Nano 33 BLE Sense и всех необходимых библиотек, выберите ее в списке плат в среде Arduino IDE (**Инструменты | Плата**), а также укажите правильный последовательный порт для ее подключения. На момент подготовки этой книги в среде Arduino IDE используются одинаковые настройки как для платы Nano 33 BLE, так и для платы Nano 33 BLE Sense (плата Nano 33 BLE точно такая же, как плата Nano 33 BLE Sense, только без датчиков). Теперь загрузите скетч из листинга 6.1 в свою плату Arduino Nano 33 BLE Sense, запустите монитор порта и наблюдайте за выводимыми в нем результатами.

Листинг 6.1. Использование датчиков платы Arduino Nano 33 BLE Sense

```
/*
* Скетч Arduino Nano BLE Sense sensor demo
*/

#include <Arduino_APDS9960.h>
#include <Arduino_HTS221.h>
#include <Arduino_LPS22HB.h>
#include <Arduino_LSM9DS1.h>

void setup()
{
   Serial.begin(9600);
   while (!Serial);
   if (!APDS.begin())
   {
      // Инициализируем датчик определения жестов, цвета и приближения
      Serial.println("Could not initialize APDS9960.");
                  // Не удалось инициализировать датчик APDS9960
      while (1);
   }
   if (!HTS.begin())
   {
      // Инициализируем датчик температуры и влажности
      Serial.println("Could not initialize HTS221.");
                  // Не удалось инициализировать датчик HTS221
      while (1);
   }
   if (!BARO.begin())
   {
      // Инициализируем барометр
      Serial.println("Could not initialize LPS22HB.");
                  // Не удалось инициализировать датчик LPS22HB
      while (1);
   }
```

```
  if (!IMU.begin())
  {
    // Инициализируем блок инерциальных измерений
    Serial.println("Could not initialize LSM9DS1.");
                    // Не удалось инициализировать датчик LSM9DS1
    while (1);
  }
  prompt(); // Сообщаем пользователям, что они могут делать
}

void loop()
{

  // При обнаружении жеста исполняем соответствующую функцию
  if (APDS.gestureAvailable())
  {
    int gesture = APDS.readGesture();
    switch (gesture)
    {
      case GESTURE_UP:
        readTemperature();
        break;

      case GESTURE_DOWN:
        readHumidity();
        break;

      case GESTURE_LEFT:
        readPressure();
        break;

      case GESTURE_RIGHT:
        Serial.println("Spin the gyro!\nx, y, z");
                        // Вращайте гироскоп
        for (int i = 0; i < 10; i++)
        {
          readGyro();
          delay(250);
        }
        break;

      default:
        break;
    }
    prompt(); // Снова отображаем меню возможностей
  }
}
```

```
void prompt()
{
   Serial.println("\nSwipe!"); // Проведите открытой ладонью
   Serial.println("Up for temperature, down for humidity");
                  // Вверх, чтобы измерить температуру;
                  // вниз, чтобы измерить относительную влажность
   Serial.println("Left for pressure, right for gyro fun.\n");
                 // Влево, чтобы измерить атмосферное давление;
                 // вправо, чтобы поиграть с гироскопом
}

void readTemperature()
{
   float temperature = HTS.readTemperature(FAHRENHEIT);
   Serial.print("Temperature: "); Serial.print(temperature);
   Serial.println(" °F");
}

void readHumidity()
{
   float humidity = HTS.readHumidity();
   Serial.print("Humidity: "); Serial.print(humidity);
   Serial.println(" %");
}

void readPressure()
{
   float pressure = BARO.readPressure(PSI);
   Serial.print("Pressure: "); Serial.print(pressure);
   Serial.println(" psi");
}

void readGyro()
{
   float x, y, z;
   if (IMU.gyroscopeAvailable())
   {
      IMU.readGyroscope(x, y, z);
      Serial.print(x); Serial.print(", ");
      Serial.print(y); Serial.print(", ");
      Serial.println(z);
   }
}
```

После загрузки скетча в мониторе порта отобразится краткая инструкция по взаимодействию с платой Arduino Nano 33 BLE Sense для использования возможностей ее датчиков. Чтобы выбрать требуемое измерение, разместите свою руку с откры-

той ладонью поверх платы и проведите ею в требуемом направлении. Движением вверх считается направление от разъема USB платы к модулю u-blox на противоположном ее конце.

Обсуждение работы решения и возможных проблем

Код скетча в листинге 6.1 использует несколько датчиков, встроенных в плату Nano 33 BLE Sense: датчик жестов (APDS-9960), датчик температуры и относительной влажности (HTS221), барометр (LPS22HB) и гироскоп (LSM9DS1). Функция `setup()` ожидает, пока не будет открыт монитор порта, а затем выполняет инициализацию всех этих устройств. В случае проблем с инициализацией какого-либо из них выводится соответствующее сообщение и скетч останавливает дальнейшее исполнение, входя в бесконечный цикл `while(1);`. В конце функции `setup()` скетч вызывает функцию отображения инструкций для пользователя в окне монитора порта.

В функции `loop()` выполняется проверка на обнаружение движения (жеста) датчиком APDS-9960. При положительном результате проверки вызывается функция для обработки данных с соответствующего датчика. Все эти функции считывают состояние своего датчика и отображают полученный результат в окне монитора порта. Для датчика гироскопа скетч отображает инструкцию по вращению платы в разных плоскостях, а затем начинает исполнение цикла, в котором показания гироскопа считываются 10 раз с небольшими интервалами, чтобы можно было видеть, как изменяются значения в результате вращения гироскопа.

Дополнительная информация

Компания Arduino поддерживает веб-страницу форума по плате Nano 33 BLE Sense (**https://oreil.ly/K7GoZ**). Также может быть полезным посетить веб-страницу форума по плате Nano 33 BLE (**https://oreil.ly/BGb2v**), которая представляет собой вариант платы без датчиков.

Дополнительная информация по использованию гироскопа с платой Arduino приведена в *разд. 6.15*.

В *разд. 6.17* приводится дополнительная информация по работе с акселерометрами.

6.2. Определение движений

ЗАДАЧА

Требуется определять перемещение, наклон или вибрацию какого-либо объекта.

РЕШЕНИЕ

Эта задача решается подключением к плате Arduino переключателя, контакты которого замыкаются при его наклоне. Такой переключатель называется *датчиком наклона*. Для работы с датчиком наклона можно использовать решения из *разд. 5.1* и *5.2*, подключив к плате Arduino датчик наклона вместо кнопки.

А в этом решении при наклоне датчика в одну сторону включается светодиод, подключенный к контакту 11, а при наклоне в другую — подключенный к контакту 12. Схема подключения светодиодов и датчика наклона к плате Arduino показана на рис. 6.1, а в листинге 6.2 приводится код скетча для работы с этой схемой.

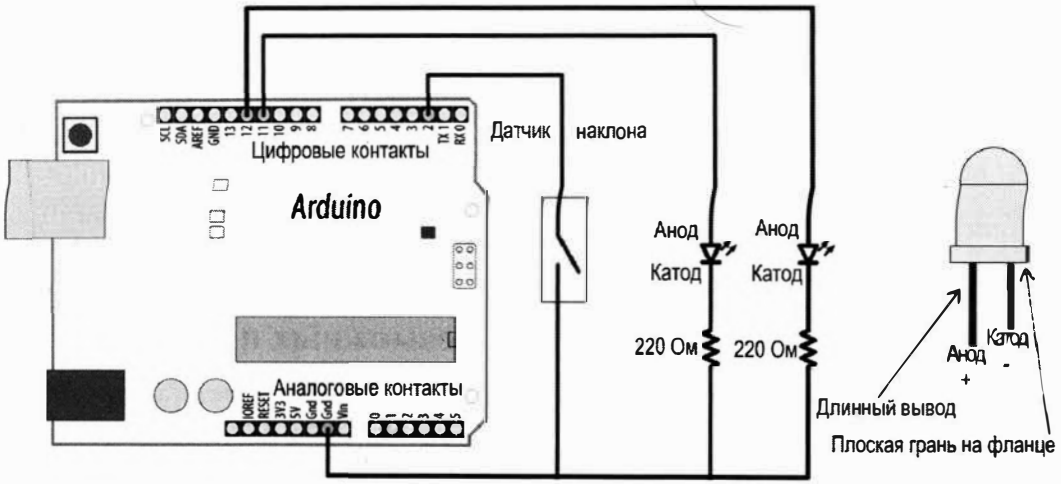


Рис. 6.1. Подключение к плате Arduino датчика наклона и двух светодиодов

Листинг 6.2. Управление светодиодами с помощью датчика наклона

```
/*
* Скетч tilt
*
* Датчик наклона включает один из двух светодиодов — в зависимости
  от стороны наклона
*/

const int tiltSensorPin = 2;  // Номер контакта для подключения датчика наклона
const int firstLEDPin = 11;   // Номер контакта для подключения первого светодиода
const int secondLEDPin = 12;  // Номер контакта для подключения второго светодиода

void setup()
{
   pinMode (tiltSensorPin, INPUT_PULLUP); // Задаем режим ввода
                                          // для контакта датчика наклона
   pinMode (firstLEDPin, OUTPUT);         // Задаем режим вывода
                                          // для контакта первого светодиода
   pinMode (secondLEDPin, OUTPUT);        // Задаем режим вывода
                                          // для контакта второго светодиода
}

void loop()
{
```

```
    if (digitalRead(tiltSensorPin) == LOW)
    {
        // Контакты датчика замкнуты (датчик в вертикальном положении)
        digitalWrite(firstLEDPin, HIGH); // Включаем первый светодиод,
        digitalWrite(secondLEDPin, LOW); // и выключаем второй.
    }
    else
    {
        // Контакты датчика разомкнуты (датчик наклонен)
        digitalWrite(firstLEDPin, LOW);   // Выключаем первый светодиод,
        digitalWrite(secondLEDPin, HIGH); // и включаем второй.
    }
}
```

Обсуждение работы решения и возможных проблем

Наиболее распространенная конструкция датчика наклона представляет собой трубку с контактами на одном конце, в которую вставлен металлический шарик чуть меньшего диаметра, чем внутренний диаметр трубки. При наклоне трубки шарик откатывается от контактов, разрывая соединение. А при наклоне трубки в другую сторону шарик подкатывается к контактам, замыкая цепь. Требуемая ориентация датчика указывается маркировкой на его корпусе. Когда датчик наклона находится в вертикальном положении с контактами, замкнутыми шариком, он способен определять незначительные отклонения от вертикали величиной приблизительно от 5 до 10 градусов. Если датчик наклона расположить таким образом, чтобы шарик находился на противоположном от контактов конце трубки, то замыкание контактов произойдет только при переворачивании датчика. Это поведение можно использовать для определения двух противоположных вертикальных положений объекта (т. е. в нормальном положении и в положении, перевернутом на 180 градусов).

Объект, оснащенный датчиком наклона, можно проверить на наличие вибрации. Для этого нужно измерить длительность периода, прошедшего после последнего изменения состояния контактов датчика (в решении, приведенном в листинге 6.2, выполняется только проверка на замкнутость или разомкнутость контактов). Если состояние контактов не изменилось в течение заданного периода времени, объект не вибрирует. Интенсивность вибрации, необходимая для активирования датчика наклона, зависит от его ориентации. В листинге 6.3 приводится код скетча, который включает встроенный светодиод платы Arduino при наличии вибрации датчика наклона.

Листинг 6.3. Определение вибрации датчика наклона

```
/*
* Скетч shaken
* Определяет вибрацию датчика наклона, включая встроенный светодиод
*/
```

```
const int tiltSensorPin = 2;
const int ledPin = LED_BUILTIN;

int tiltSensorPreviousValue = 0;
int tiltSensorCurrentValue = 0;
long lastTimeMoved = 0;
int shakeTime = 50;

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

void loop()
{
   tiltSensorCurrentValue = digitalRead(tiltSensorPin);
   if (tiltSensorPreviousValue != tiltSensorCurrentValue)
   {
      lastTimeMoved = millis();
      tiltSensorPreviousValue = tiltSensorCurrentValue;
   }
   if (millis() - lastTimeMoved < shakeTime)
   {
      digitalWrite(ledPin, HIGH);
   }
   else
   {
      digitalWrite(ledPin, LOW);
   }
}
```

Подобным образом можно использовать многие механические контактные датчики. Например, датчик уровня воды включается, когда уровень воды в контейнере поднимается до определенного уровня (подобно тому, как работает плавающий клапан в сливном бачке туалета). А тензодатчик можно использовать для определения момента, когда на него наступили (как это делается в некоторых магазинах, чтобы известить персонал о приходе клиента). Если ваш датчик выдает простой цифровой сигнал включенного и выключенного уровня, то для работы с ним можно использовать скетч наподобие показанного в листинге 6.2.

Дополнительная информация

В *главе 5* приводится дополнительная справочная информация по использованию кнопочных переключателей с платформой Arduino. Дополнительная информация по использованию функции `millis()` также приводится в *разд. 12.1*.

6.3. Определение уровня освещенности

ЗАДАЧА

Требуется определять изменение уровня освещенности. Например, зафиксировать момент прохождения объекта перед фотодатчиком или измерить уровень освещенности в комнате для управления системой освещения.

РЕШЕНИЕ

Проще всего измерять уровень освещенности с помощью фоторезистора. Сопротивление фоторезистора меняется в зависимости от его освещенности, в результате чего меняется падение напряжение на нем. Подключив его к плате Arduino, как показано на рис. 6.2, можно измерять это падение напряжения и сопоставлять его уровню освещенности.

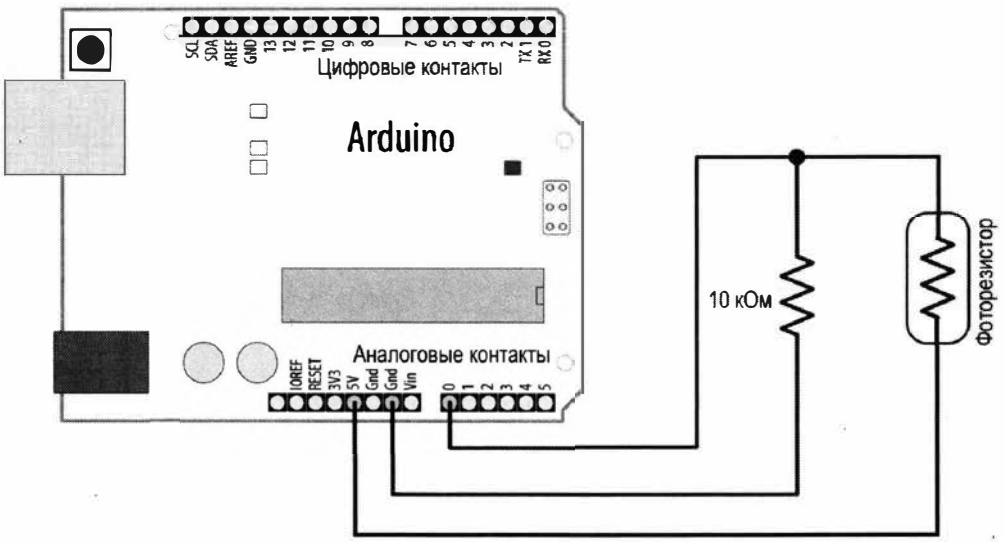


Рис. 6.2. Подключения фоторезистора к плате Arduino

В листинге 6.4 приводится скетч для обработки выходного сигнала фоторезистора.

Листинг 6.4. Обработка выходного сигнала фоторезистора

```
/*
* Скетч Light sensor
*
* Меняет частоту мигания встроенного светодиода в зависимости от уровня освещенности
*/

const int ledPin = LED_BUILTIN; // Константа для номера контакта
                                // встроенного светодиода
```

```
const int sensorPin = A0;       // Подключаем выход светодатчика
                                // к контакту 0 аналогового ввода

void setup()
{
   pinMode(ledPin, OUTPUT); // Задаем выходной режим работы
                            // для контакта подключения светодиода
}

void loop()
{
   int rate = analogRead(sensorPin); // Считываем входной аналоговый сигнал
   digitalWrite(ledPin, HIGH);       // Включаем встроенный светодиод
   delay(rate);                      // Длительность задержки задается
                                     // уровнем освещения
   digitalWrite(ledPin, LOW);        // Выключаем встроенный светодиод
   delay(rate);
}
```

Фоторезисторы содержат опасное соединение — сульфид кадмия, поэтому они запрещены в некоторых странах. Фоторезистор можно без каких бы то ни было проблем заменить фототранзистором. Подобно светодиоду, фототранзистор имеет один вывод более длинный, чем другой. Вы можете подключить фототранзистор по схеме, показанной на рис. 6.2 для фоторезистора, но его длинный вывод нужно подключить к шине питания 5 В, а короткий — к резистору и контакту 0. Разные фототранзисторы реагируют на разные участки оптического спектра. Нам нужен фототранзистор, реагирующий на видимый свет (а не на инфракрасный), — например, фототранзистор компании Adafruit артикул 2831 (https://oreil.ly/tUa__).

Обсуждение работы решения и возможных проблем

В этом решении представлен стандартный способ подключения любого датчика, сопротивление которого меняется под воздействием какого-либо физического явления (справочная информация по работе с аналоговыми сигналами приведена в *главе 5*). В схеме, показанной на рис. 6.2, напряжение на аналоговом контакте 0 меняется в результате изменения сопротивления фоторезистора (или фототранзистора) под воздействием разных уровней освещенности.

Эта схема не позволит получить полный диапазон возможных значений для входного аналогового сигнала (от 0 до 1023), поскольку напряжение этого сигнала не занимает диапазон от 0 до 5 вольт. Дело в том, что на каждом сопротивлении (резистора и фоторезистора) всегда будет присутствовать падение напряжения, поэтому напряжение в точке их соединения никогда не достигнет пределов напряжения источника питания. Работая с датчиками, наподобие задействованного в этом решении, важно проверить фактические возвращаемые датчиком значения в ситуации, в которой он будет использоваться. Затем нужно определить, как преобразо-

вать эти значения в значения, необходимые для управления требуемым устройством. Более подробная информация по масштабированию значения одного диапазона в другой приводится в *разд. 5.7*.

Фоторезистор принадлежит к типу простых датчиков, называющихся *резистивными датчиками*. Ряд резистивных датчиков реагируют на изменения разных физических характеристик.

Поскольку плата Arduino не может измерять непосредственно сопротивление, в решении используется постоянный резистор, который в комбинации с резистивным датчиком (фоторезистором) составляет делитель напряжения, наподобие рассмотренного в *разд. 5.11*. Аналоговые контакты платы Arduino реагируют на напряжения, а не на сопротивление, поэтому для того чтобы плата Arduino могла измерить сопротивление, необходимо, чтобы это сопротивление каким-либо образом меняло напряжение. Величина напряжения на выходе делителя напряжения из двух резисторов зависит от входного напряжения и сопротивления каждого из резисторов делителя. Таким образом, если один из резисторов делителя напряжения имеет постоянное сопротивление, а сопротивление другого может варьироваться — как, например, сопротивление фоторезистора, на аналоговый контакт платы Arduino станет поступать аналоговый сигнал, напряжение которого будет меняться, реагируя на изменения сопротивления фоторезистора.

Подобные схемы можно использовать с другими типами простых резистивных датчиков, хотя может потребоваться откорректировать сопротивление постоянного резистора в зависимости от конкретного датчика. Выбор значения сопротивления постоянного резистора делителя зависит от типа используемого фоторезистора и диапазона уровней освещения, которые требуется отслеживать. Для решения этой задачи инженеры использовали бы люксметр и технические данные фоторезистора с его справочного листка. Но эту задачу можно решить более простым способом с помощью мультиметра. Установите мультиметр на измерение сопротивления и измерьте сопротивление фоторезистора при уровне освещенности, равном приблизительно половине диапазона уровней освещенности, в котором вы планируете его использовать. Сопротивление постоянного резистора должно быть ближайшего удобного номинала к измеренному значению. Сопротивление фоторезистора при разных уровнях освещенности можно также считывать с помощью скетча Arduino и выводить эти значения в окно монитора порта или же в окно плоттера по последовательному соединению для графического отображения. Подробная информация по выводу данных в окно монитора порта и плоттера приведена в *разд. 4.1*.

Следует иметь в виду, что любые источники искусственного освещения, включающиеся и выключающиеся в ходе работы с очень высокой частотой, например неоновые лампы или некоторые светодиодные лампы, могут воздействовать на показания датчика. Хотя их частота включения-выключения неразличима человеческим глазом, для Arduino их уровень освещения может казаться низким. Чтобы внести поправку на этот эффект, можно взять среднее значение показаний, пример вычисления которого приводится в *разд. 6.8*.

Дополнительная информация

Скетч рассматриваемого здесь решения впервые приводился в *разд. 1.6*. Обратитесь к нему за дополнительной информацией по этому скетчу и его разновидностям.

6.4. Определение движения живых существ

ЗАДАЧА

Требуется определять движение вблизи датчика людей или животных.

РЕШЕНИЕ

Пассивный инфракрасный (ПИК)[4] датчик движения определяет наличие в зоне своего действия живых существ (или объектов, излучающих тепло), выдавая соответствующий сигнал, который можно подать на аналоговый контакт платы Arduino для дальнейшей обработки.

В качестве такого датчика можно использовать датчик компании Adafruit (артикул 189) или датчик компании Parallax (артикул 555-28027), которые подключаются к плате Arduino, как показано на рис. 6.3. Некоторые ПИК-датчики — например, датчик PIR Motion Sensor компании SparkFun (артикул SEN-13285), требуют подключения повышающего резистора к выходу датчика. В случае использования таких датчиков необходимо задать входной режим работы для контакта их подклю-

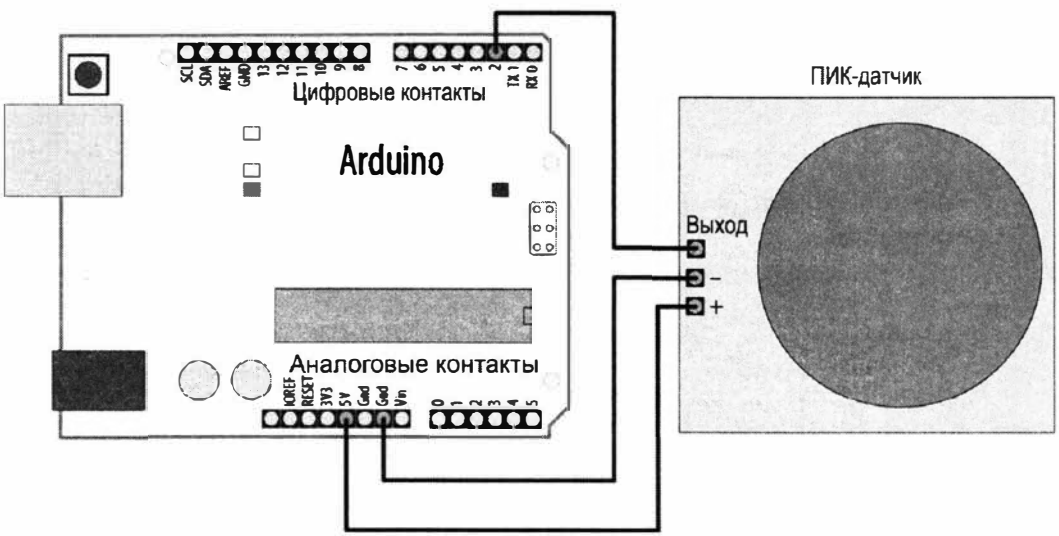


Рис. 6.3. Подключение ПИК-датчика движения к плате Arduino

[4] ПИК — от *англ.* PIR, Passive Infrared.

чения, используя параметр INPUT_PULLUP, а также инвертировать логику скетча, как рассматривается далее в *разд. «Обсуждение работы решения и возможных проблем»*.

При подключении своего датчика к плате Arduino, загляните в справочный листок на него, чтобы определить правильные выводы. Например, выводы датчика компании Adafruit обозначены OUT, – и + (что означает выход, «земля» и +5 В), а выводы датчика компании Parallax обозначены GND, VCC и OUT («земля», +5 В и выход).

В листинге 6.5 приводится код скетча, который включает встроенный светодиод при обнаружении датчиком какого-либо движения.

Листинг 6.5. Обнаружение движения с помощью ПИК-датчика

```
/*
Скетч PIR
Пассивный инфракрасный датчик, подключенный к контакту 2,
включает встроенный светодиод
*/

const int ledPin = LED_BUILTIN; // Контакт для подключения светодиода
const int inputPin = 2;         // Контакт для подачи входного сигнала
                                // (с ПИК-датчика движения)

void setup()
{
   pinMode(ledPin, OUTPUT);  // Задаем выходной режим работы для контакта светодиода
   pinMode(inputPin, INPUT); // Задаем входной режим работы для контакта датчика
}

void loop()
{
   int val = digitalRead(inputPin); // Считываем входное значение
   if (val == HIGH)                 // Проверяем наличие высокого
                                    // входного уровня на контакте
   {
      digitalWrite(ledPin, HIGH);   // Включаем светодиод при определении движения
      delay(500);
      digitalWrite(ledPin, LOW); // Выключаем светодиод
   }
}
```

Обсуждение работы решения и возможных проблем

Код этого решения похож на код примеров работы с кнопкой, рассматриваемых в *главе 5*. Это объясняется тем, что при обнаружении движения датчик движения реагирует подобно кнопке. Но на рынке имеются разные типы ПИК-датчиков дви-

жения, и для подключения и работы со своим датчиком нужно разобраться с его рабочими характеристиками.

Некоторые датчики — например, датчики компаний Parallax и Adafruit, оснащены перемычками для выбора типа выходного сигнала при обнаружении движения. В одном режиме при обнаружении движения уровень выходного сигнала остается высоким, а в другом уровень становится высоким на короткое время, а затем переходит на низкий. Скетч приведенного в листинге 6.5 решения будет работать с любым режимом датчика.

Другие датчики при обнаружении движения выдают сигнал низкого уровня. Если ваш датчик выдает сигнал низкого уровня при обнаружении движения, в скетче решения измените строку кода, в которой выполняется проверка сигнала, подаваемого с датчика, для включения светодиода при низком (LOW) уровне:

```
if (val == LOW) // Обнаруженное движение указывается низким (LOW) уровнем
```

Если на выходном контакте вашего датчика необходимо установить повышающий резистор, строку кода для инициализации контакта ввода `inputPin` необходимо заменить следующим кодом:

```
pinMode(inputPin, INPUT_PULLUP); // Задаем входной режим работы контакта
                                 // и включаем встроенный повышающий резистор
```

На рынке доступны пассивные инфракрасные датчики движения разных видов, с разными диапазонами и углами чувствительности. Правильно выбранный и установленный датчик будет реагировать на движение в требуемой части комнаты, а не во всей комнате. Некоторые ПИК-датчики оснащены подстроечным потенциометром, посредством которого можно откорректировать чувствительность датчика.

6.5. Измерение расстояния

ЗАДАЧА

Требуется измерить расстояние до какого-либо предмета — как неподвижного, так и приближающегося.

РЕШЕНИЕ

Эта задача решается с помощью ультразвукового датчика расстояния PING))) компании Parallax, который может измерять расстояние до объектов в диапазоне от 2 сантиметров до приблизительно 3 метров. Подключение датчика к плате Arduino показано на рис. 6.4. Скетч решения (листинг 6.6) обрабатывает получаемый с датчика сигнал и выводит расстояние в окне монитора порта, а также мигает встроенным светодиодом. При этом чем ближе объект приближается к датчику, тем частота мигания светодиода повышается.

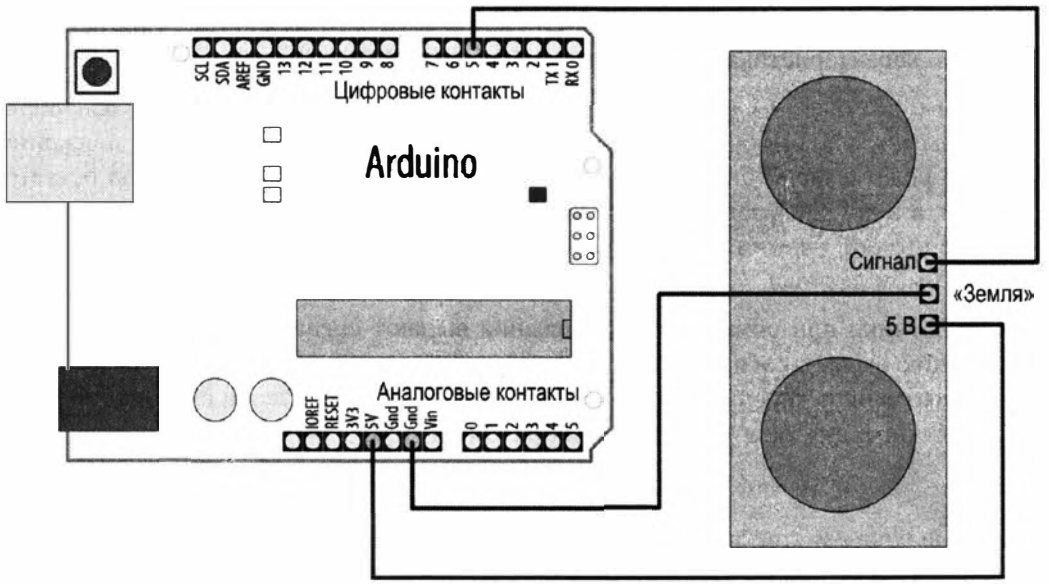


Рис. 6.4. Подключение ультразвукового датчика расстояния Ping))) к плате Arduino

Листинг 6.6. Определение расстояние до объекта

```
/* Скетч Ping))) Sensor
* Выводит расстояние до объекта в окне монитора порта и мигает
  встроенным светодиодом с частотой, пропорциональной расстоянию до объекта
*/

const int pingPin = 5;
const int ledPin = LED_BUILTIN; // Константа номера контакта встроенного светодиода

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

void loop()
{
   int cm = ping(pingPin);
   Serial.println(cm);
   digitalWrite(ledPin, HIGH);
   delay(cm * 10); // Каждый сантиметр добавляет 10 мс задержки
   digitalWrite(ledPin, LOW);
   delay(cm * 10);
}
```

```
// Измеряем расстояние в сантиметрах и возвращаем результат
int ping(int pingPin)
{
   long duration; // Переменная для хранения измеренной
                  // длительности импульса

   // Задаем выходной режим работы для контакта встроенного светодиода
   pinMode(pingPin, OUTPUT);
   digitalWrite(pingPin, LOW); // Удерживаем низкий уровень
        // в течение 2 мкс, чтобы обеспечить чистый импульс
   delayMicroseconds(2);

   // Посылаем импульс длительностью 5 мкс
   digitalWrite(pingPin, HIGH);
   delayMicroseconds(5);
   digitalWrite(pingPin, LOW);

   // Задаем входной режим работы для контакта датчика и считываем
   // поступающий с датчика сигнал
   pinMode(pingPin, INPUT);
   duration = pulseIn(pingPin, HIGH);

   // Преобразовываем время в расстояние
   return duration / 29 / 2;
}
```

Обсуждение работы решения и возможных проблем

Ультразвуковые датчики расстояния измеряют время прохождения ультразвукового импульса до объекта и обратно — после отражения от объекта.

Ультразвуковой импульс создается, когда на контакте `pingPin` в течение двух микросекунд удерживается сигнал высокого уровня. Генерирование импульса прекращается, когда возвращается отраженный от объекта сигнал. Таким образом, длительность импульса прямо пропорциональна расстоянию, проходимому им до объекта и обратно, и измеряется в скетче с помощью функции `pulseIn()`. Скорость звука составляет приблизительно 340 метров в секунду, или 1 сантиметр за 29 микросекунд. Таким образом, двойное расстояние (в сантиметрах) до объекта вычисляется делением длительности импульса на 29 микросекунд.

А собственно расстояние до объекта получается делением этого результата на 2. Скорость звука в 340 метров в секунду наблюдается при температуре 20°C. Если в вашем случае температура окружающей среды значительно отличается от этого значения, узнать соответствующую ей скорость звука можно с помощью какого-либо онлайнового калькулятора этой величины — например, калькулятора на вебсайте Национальной погодной службы США (**https://oreil.ly/a9KXK**).

Менее дорогой альтернативой датчику PING))) компании Parallax может служить датчик HC-SR04, который предлагается многими поставщиками, а также на веб-

площадке eBay. Хотя этот датчик менее точный и имеет меньший радиус действия, его можно использовать в тех случаях, когда дешевизна более важна, чем производительность. В датчике HC-SR04 для запуска ультразвукового импульса и фиксации его возвращения используются отдельные выводы. Схема подключения этого датчика к плате Arduino показана на рис. 6.5, а в листинге 6.7 приводится скетч для работы с ним.

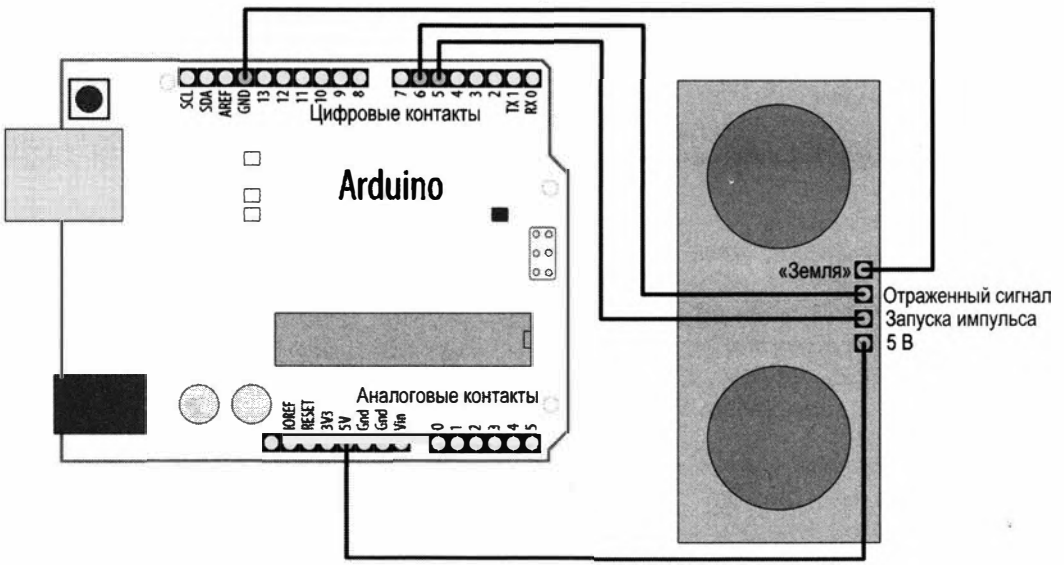


Рис. 6.5. Подключение датчика расстояния HC-SR04 к плате Arduino

Листинг 6.7. Скетч для работы с датчиком HC-SR04

```
/* Скетч HC-SR04 Sensor
* Выводит расстояние до объекта в окне монитора порта и мигает
  встроенным светодиодом с частотой, пропорциональной расстоянию до объекта
*/

const int trigPin = 5;    // Контакт для подачи сигнала запуска импульса
const int echoPin = 6;    // Контакт для считывания сигнала датчика
const int ledPin = LED_BUILTIN; // Константа номера контакта встроенного светодиода

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

void loop()
{
```

```
    int cm = calculateDistance(trigPin);
    Serial.println(cm);

    digitalWrite(ledPin, HIGH);
    delay(cm * 10);      // Каждый сантиметр добавляет 10 мс задержки
    digitalWrite(ledPin, LOW);
    delay(cm * 10);

    delay(60); // Справочный листок рекомендует выдерживать паузу
               // между измерениями длительностью минимум 60 мс
}

int calculateDistance(int trigPin)
{
    long duration; // Переменная для хранения измеренной длительности импульса

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2); // Удерживаем низкий уровень
            // в течение 2 мкс, чтобы обеспечить чистый импульс
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10); // Посылаем импульс длительностью 10 мкс
    digitalWrite(trigPin, LOW);

    // Считываем длительность импульса
    duration = pulseIn(echoPin, HIGH);

    // Преобразовываем время в расстояние
    return duration / 29 / 2;
}
```

В справочном листке (datasheet) для датчика HC-SR04 рекомендуется выжидать по крайней мере 60 мс между измерениями, но мигание светодиодом также занимает некоторое время, поэтому команда `delay(60);` добавляет бóльшую задержку, чем требуется. Но если разрабатываемый вами код не вносит никаких собственных задержек, следует оставить минимальную задержку в 60 мс.

Датчик расстояния HC-SR04 лучше всего использовать с платами с рабочим напряжением 5 В, но он также может взаимодействовать и с платами с рабочим напряжением 3,3 В, которые способны принимать логические сигналы напряжением 5 В, как, например, плата Teensy 3.

Еще одним ультразвуковым датчиком, который можно применить для измерения расстояния, является датчик EZ1 компании MaxBotix. С этим датчиком легче работать, чем с датчиком Ping))) или HC-SR04, поскольку для него не нужно подавать сигнал запуска импульса, а также он может работать с напряжением питания как 5 В, так и 3,3 В. Он постоянно выдает информацию о расстоянии или в виде аналогового сигнала, или виде ШИМ-сигнала (сигнала с широтно-импульсной модуляцией).

Подключение датчика EZ1 к плате Arduino показано на рис. 6.6. Питание на модуль датчика подается на его контакты +5V и Gnd, так что подключите эти контакты к соответствующим контактам платы Arduino. Линию выходного ШИМ-сигнала датчика EZ1 подключите к контакту 5 платы Arduino.

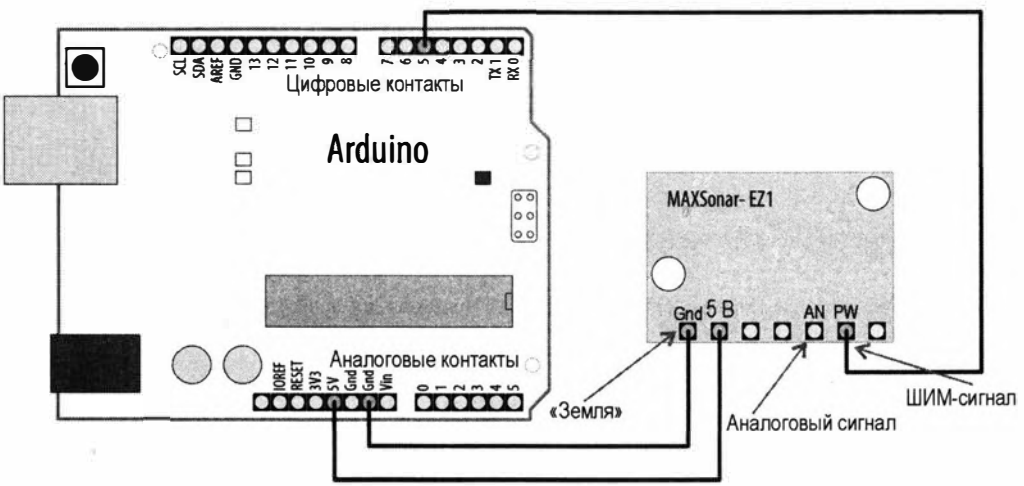


Рис. 6.6. Подключение датчика расстояния MaxBotix EZ1 к плате Arduino

В листинге 6.8 приводится скетч для работы с датчиком EZ1 компании MaxBotix. В скетче используется длительность импульсов выходного ШИМ-сигнала датчика, создающая информацию о расстоянии в формате, аналогичном формату, создаваемому в предыдущем скетче.

Листинг 6.8. Обработка выходного ШИМ-сигнала датчика EZ1 компании MaxBotix

```
/*
* Скетч EZ1Rangefinder Distance Sensor
* Выводит расстояние до объекта в окне монитора порта и мигает
  встроенным светодиодом с частотой, пропорциональной расстоянию до объекта
*/

const int sensorPin = 5;
const int ledPin = LED_BUILTIN;

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

void loop()
{
   long value = pulseIn(sensorPin, HIGH) ;
```

```
    int cm = value / 58; // Длительность импульса составляет 58 мкс/см
    Serial.println(cm);

    digitalWrite(ledPin, HIGH);
    delay(cm * 10 ); // Каждый сантиметр добавляет 10 мс задержки
    digitalWrite(ledPin, LOW);
    delay( cm * 10);

    delay(20);
}
```

Скетч решения измеряет длительность импульса посредством команды `pulseIn()`. Длительность импульса составляет 58 микросекунд на сантиметр, или 147 микросекунд на дюйм.

В случае использования длинных монтажных проводов может потребоваться подключить конденсатор к линии питания (между шиной +5 В и шиной «земли»), чтобы стабилизировать питание датчика. При получении непостоянных показаний датчика подключите на его линию питания конденсатор емкостью 10 мкФ (в *приложении 3* приводится дополнительная информация по использованию развязывающих конденсаторов).

Показания датчика EZ1 можно также получать с его аналогового выхода, подключив контакт AN датчика к аналоговому контакту платы Arduino и считывая значение посредством функции `analogRead()`. В листинге 6.9 приводится фрагмент кода для получения показаний датчика с его выходного аналогового сигнала.

Листинг 6.9. Получение показаний датчика EZ1 с выходного аналогового сигнала

```
int value = analogRead(A0);
float mv = (value / 1024.0) * 5000 ;
float inches = mv / 9.8; // Согласно справочному листку 9,8 мВ/дюйм
float cm = inches * 2.54;
Serial.print("in: "); Serial.println(inches);
Serial.print("cm: "); Serial.println(cm);
```

Каждая единица значения, возвращаемого функцией `analogRead()`, соответствует приблизительно 4,8 мВ входящего аналогового сигнала (дополнительная информация по работе с функцией `analogRead()` приводится в *разд. 5.6*), и согласно справочному листку каждому дюйму расстояния соответствует 9,8 мВ выходного сигнала датчика EZ1 при питании 5 В или 6,4 мВ при питании 3,3 В. Чтобы получить расстояния в сантиметрах, результат в дюймах нужно умножить на 2,54.

Дополнительная информация

Преобразование числовых значений, возвращаемых функцией `analogRead()`, в соответствующие значения напряжения рассматривается в *разд. 5.6*. Более подробная информация по функции `pulseIn()` приводится на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/F6fdU**).

6.6. Точное измерение расстояния

ЗАДАЧА

Требуется измерять расстояние до объектов с большей точностью, чем позволяют решения, приведенные в *разд. 6.5*.

РЕШЕНИЕ

Датчики расстояния ToF[5] работают по принципу инфракрасных и ультразвуковых датчиков расстояния, измеряя время прохождения излучаемого сигнала до объекта и обратно, но в качестве сигнала используют лазерный луч. Эти датчики имеют намного более узкое поле зрения, чем ультразвуковые датчики, рассматриваемые в *разд. 6.5*, и возвращают более точные показания. Однако у датчиков ToF обычно более короткий диапазон измеряемых расстояний. Например, если диапазон измерений датчика HC-SR04 составляет от 2 см до 4 метров, ToF-датчик расстояния VL6180X компании Adafruit (артикул 3316) имеет диапазон измерений от 5 до 10 см.

В листинге 6.10 приводится код скетча, предоставляющего функциональность, подобную функциональности скетча решения из *разд. 6.5*, но с использованием ToF-датчика расстояния VL6180X. Подключение датчика VL6180X к плате Arduino показано на рис. 6.7. Для работы скетча необходимо установить библиотеку Adafruit_VL6180X (подробная информация по установке библиотек приводится в *разд. 16.2*).

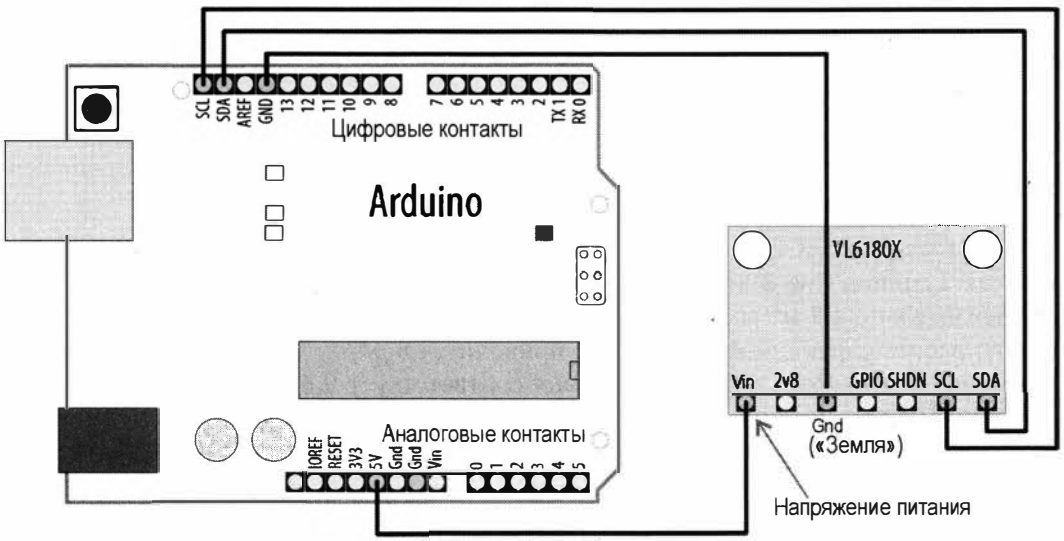


Рис. 6.7. Подключение ToF-датчика расстояния VL6180X к плате Arduino

[5] ToF — от *англ.* Time of Flight, время прохождения (здесь — фотонами) расстояния до объекта.

Листинг 6.10. Измерение расстояния с использованием ToF-датчика VL6180X

```
/* Скетч tof-distance
* Выводит расстояние до объекта в окне монитора порта и мигает
  встроенным светодиодом с частотой, пропорциональной расстоянию до объекта
*/

#include <Wire.h>
#include "Adafruit_VL6180X.h"

Adafruit_VL6180X sensor = Adafruit_VL6180X();

const int ledPin = LED_BUILTIN; // Константа номера контакта встроенного светодиода

void setup()
{
   Serial.begin(9600);
   while (!Serial);
   if (! sensor.begin())
   {
      Serial.println("Could not initialize VL6180X");
                     // Не удалось инициализировать датчик VL6180X
      while (1);
   }
}

void loop()
{
   // Считываем расстояние и проверяем статус на наличие ошибок
   byte cm = sensor.readRange();
   byte status = sensor.readRangeStatus();
   if (status == VL6180X_ERROR_NONE)
   {
      Serial.println(cm);

      digitalWrite(ledPin, HIGH);
      delay(cm * 10); // Каждый сантиметр добавляет 10 мс задержки
      digitalWrite(ledPin, LOW);

      delay(cm * 10);
   }
   else
   {
      // На серьезные ошибки следует обратить внимание
      if ((status >= VL6180X_ERROR_SYSERR_1) &&
      (status <= VL6180X_ERROR_SYSERR_5))
```

```
        {
            Serial.println("System error"); // Системная ошибка
        }
    }
    delay(50);
}
```

Обсуждение работы решения и возможных проблем

Датчик VL6180X предоставляет результаты измерений по протоколу I^2C (jy подробно рассматривается в *главе 13*), вследствие чего для его подключения к плате Arduino используются соответствующие контакты SCL и SDA датчика и платы (см. рис. 6.7). В скетче выполняется подключение библиотеки Wire, обеспечивающей поддержку протокола I^2C, а также библиотеки Adafruit_VL6180X, предоставляющей функции для работы с датчиком. В начале скетча перед функцией `setup()` определяется объект `sensor` для представления датчика, который дальше инициализируется в функции `setup()`.

Функция `setup()` инициализирует последовательный порт и предпринимает попытку инициализировать объект `sensor`. В случае неуспеха инициализации объекта `sensor` в окно монитора порта выводится соответствующее сообщение и скетч прекращает дальнейшее исполнение, входя в бесконечный цикл `while`.

В каждой итерации главного цикла `loop()` скетч считывает показания датчика, а также проверяет его статус на наличие ошибок состояния. Полученные действительные показания датчика отображаются в окне монитора порта, при этом запускается мигание встроенным светодиодом с частотой, пропорциональной измеренному расстоянию до объекта. Пример скетча, поставляемого с библиотекой Adafruit_VL6180X, выполняет более подробную проверку всех возможных состояний ошибки. Но за исключением системных ошибок, проверка на которые выполняется в приведенном здесь скетче решения, большинство этих ошибок — случайные и исправляются при последующем чтении показаний датчика.

Дополнительная информация

На веб-сайте DIY Projects (**https://oreil.ly/HmeJs**) и на веб-сайте компании SparkFun (**https://oreil.ly/b0Jc4**) приводится подробное сравнение возможностей инфракрасных, ультразвуковых и лазерных датчиков расстояния.

6.7. Определение вибраций

ЗАДАЧА

Требуется выявить наличие вибрации объекта и реагировать на нее. Например, нужно определить, когда стучатся в дверь.

РЕШЕНИЕ

Эта задача решается с помощью *пьезодатчика*, который реагирует на вибрации. Такой датчик дает наилучшие результаты, когда размещен на достаточно большой поверхности. Подключение пьезодатчика вибрации к плате Arduino показано на рис. 6.8, а в листинге 6.11 приводится скетч для работы с ним.

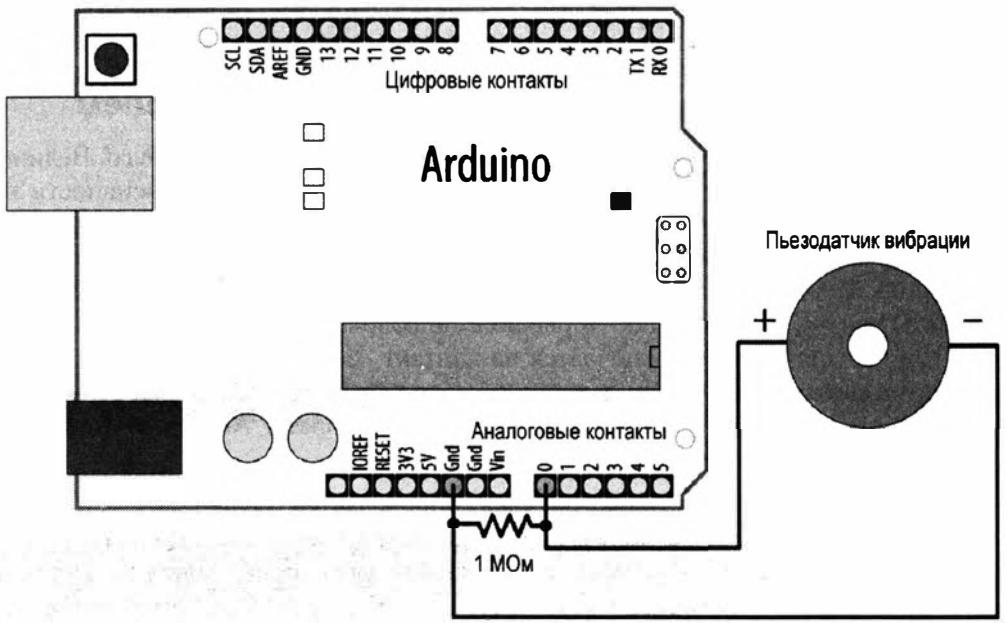


Рис. 6.8. Подключение пьезодатчика вибрации к плате Arduino

Листинг 6.11. Скетч для работы с пьезодатчиком вибрации

```
/* Скетч piezo
* Определяет вибрации и включает светодиод
*/

const int sensorPin = 0;        // Номер аналогового контакта,
                                // к которому подключается выход датчика
const int ledPin = LED_BUILTIN; // Номер цифрового контакта,
                                // к которому подключен встроенный светодиод
const int THRESHOLD = 100;

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

void loop()
{
   int val = analogRead(sensorPin);
```

```
    if (val >= THRESHOLD)
    {
       digitalWrite(ledPin, HIGH);
       delay(100); // Включаем светодиод
    }
    else
       digitalWrite(ledPin, LOW);
}
```

Обсуждение работы решения и возможных проблем

Пьезодатчик создает напряжение в ответ на физическое давление на него. Величина выходного напряжения пьезодатчика прямо пропорциональна интенсивности воздействующего на него давления. Контакты выходного сигнала датчика поляризованы, и положительный контакт (обычно с подключенным красным проводом или обозначенный знаком +) подключается к контакту аналогового ввода платы Arduino. Отрицательный контакт (обычно с подключенным черным проводом или обозначенный знаком –) подключается на контакт «земли» платы Arduino. Между контактами выходного сигнала датчика подключается резистор с высоким сопротивлением (1 МОм), чтобы предохранить контакты платы Arduino от чрезмерно высокого напряжения или тока.

Напряжение выходного сигнала датчика определяется функцией `analogRead()`, включающей встроенный светодиод в ответ на наличие активного сигнала. Уровень входного сигнала, необходимый для включения светодиода, задается значением переменной `THRESHOLD`, которое можно увеличить или уменьшить, чтобы повысить или понизить чувствительность скетча.

Пьезодатчики могут продаваться в виде металлического диска с двумя подключенными проводами или же заключенными в пластмассовый корпус. Но независимо от оформления оба типа датчиков содержат одинаковые компоненты. Используйте тот, который лучше всего подходит для вашего проекта.

Некоторыми датчиками, включая пьезодатчики, можно управлять с помощью Arduino, чтобы создавать эффект, который они могут обнаруживать. В *главе 9* приводится дополнительная информация о том, как использовать пьезодатчик для генерирования звука.

6.8. Обнаружение звука

ЗАДАЧА

Требуется обнаруживать звуки — например, разговор, крик или хлопание в ладоши.

РЕШЕНИЕ

Эта задача решается с помощью электретного микрофона, установленного на адаптерной плате (предлагается компанией SparkFun, артикул BOB-12758). Подключите

адаптерную плату с микрофоном к плате Arduino, как показано на рис. 6.9. В случае использования платы Arduino с питанием 3,3 В контакт VCC (+V) микрофона следует подключить к контакту 3V3 платы Arduino, а не к контакту 5V. Подключив микрофон, загрузите в свою плату Arduino скетч из листинга 6.12.

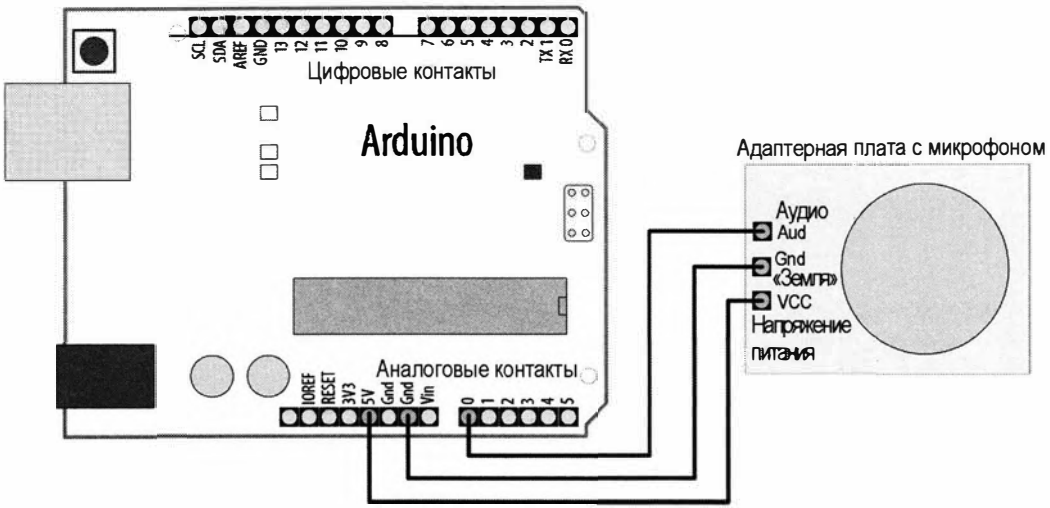


Рис. 6.9. Подключение электретного микрофона к плате Arduino

Попробуйте создать какой-либо звук — например, хлопать в ладоши, громко говорить, кричать или проигрывать громкую музыку вблизи микрофона. Скетч должен обнаруживать эти звуковые воздействия, реагируя включением встроенного светодиода. Возможно, чтобы светодиод включался, потребуется откорректировать пороговое значение переменной `threshold`. Для этого определите минимальные и максимальные показатели датчика (наблюдая их в окне монитора порта), соответствующие отсутствию и присутствию звука. Установите значение переменной `threshold` посередине полученных предельных значений. Загрузите откорректированный код и повторите эксперимент по обнаружению звука.

Листинг 6.12. Скетч для обнаружения звука с помощью электретного микрофона

```
/* Скетч microphone
* Обнаруживает звук с помощью электретного микрофона и включает
  встроенный светодиод
*/

const int micPin = A0;           // Подключаем микрофон к аналоговому контакту 0
const int ledPin = LED_BUILTIN;  // Номер контакта встроенного светодиода
const int middleValue = 512;     // Середина диапазона аналоговых значений
const int numberOfSamples = 128; // Количество выборок при каждой итерации

int sample;                  // Значение, считываемое с микрофона при каждой итерации
long signal;                 // Значение после удаления смещения
long newReading;             // Среднее значение показаний
```

```
long runningAverage = 0;      // Среднее значение
const int averagedOver = 16; // Скорость воздействия новых значений
                              // на среднее значение
                              // Чем больше значение, тем меньше скорость

const int threshold = 400;    // Пороговый уровень включения светодиода

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

void loop()
{
   long sumOfSquares = 0;
   for (int i=0; i<numberOfSamples; i++)
   {
      // Берем большое количество выборок и усредняем их значения
      sample = analogRead(micPin);      // Считываем значение сигнала микрофона
      signal = (sample - middleValue); // Вычисляем его смещение от центра
      signal *= signal;                 // Возводим его в квадрат
      sumOfSquares += signal;           // Добавляем к общему
   }

   newReading = sumOfSquares/numberOfSamples;

   // Вычисляем среднее значение
   runningAverage=(((averagedOver-1)*runningAverage)+newReading)/averagedOver;

   Serial.print("new:"); Serial.print(newReading);
   Serial.print(",");
   Serial.print("running:"); Serial.println(runningAverage);

   if (runningAverage > threshold)
   {
      // Среднее больше, чем пороговое значение?
      digitalWrite(ledPin, HIGH); // Если да, включаем светодиод
   }
   else
   {
      digitalWrite(ledPin, LOW); // Если нет, выключаем светодиод
   }
}
```

Обсуждение работы решения и возможных проблем

Микрофон создает очень слабый электрический сигнал. Если подключить его напрямую к аналоговому контакту платы Arduino, мы не сможем увидеть никаких

изменений сигнала. Чтобы Arduino мог работать с выходным сигналом микрофона, его нужно усилить. Адаптерная плата, на которой установлен микрофон, кроме него также содержит усилитель звуковой частоты, который усиливает выходной сигнал микрофона до уровня, приемлемого для использования платой Arduino.

Поскольку в этом решении мы считываем звуковой сигнал, чтобы получить из него полезную информацию, нам нужно выполнить с ним дополнительные вычисления. Звуковой сигнал изменяется весьма быстро, и возвращенное функцией `analogRead()` значение будет зависеть от того, в какой точке колеблющегося сигнала оно было считано. Если вы подзабыли, как работать с функцией `analogRead()`, вы можете освежить свои знания, обратившись к *главе 5* и *разд. 6.3*. На рис. 6.10 показана условная осциллограмма звукового сигнала. С течением времени слева направо напряжение сигнала поднимается и падает регулярным образом. Показания, взятые в трех разных точках сигнала, обозначенных на рис. 6.10 позициями 1, 2 и 3, дадут три разных значения. Если использовать эти показания для принятия решения, можно ошибочно сделать вывод, что сигнал в точке 2 сильнее, чем в других двух точках.

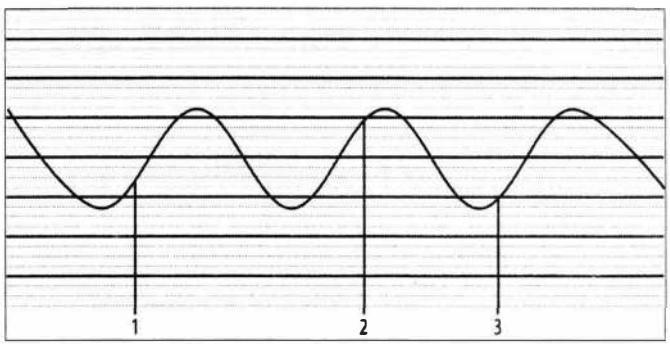


Рис. 6.10. Измерение звукового сигнала в трех разных точках

Чтобы получить точные показания, необходимо выполнить многократные замеры сигнала в близкой последовательности один за другим. По мере усиления сигнала увеличиваются его пики и впадины. Разница между минимальным и максимальным значениями сигнала называется его *амплитудой*, которая увеличивается с повышением громкости сигнала.

Чтобы измерить величину пиков и впадин сигнала, нужно измерить разницу между напряжением в средней точке сигнала и напряжениями на пиках и впадинах. Среднюю точку сигнала можно представить как линию, проходящую посередине между самым высоким пиком и самой низкой впадиной, как показано на рис. 6.11. Такая линия представляет смещение постоянной составляющей (это значение напряжения при отсутствии пиков и впадин). Отнимая значение смещения постоянной составляющей от возвращенного функцией `analogRead()` значения, мы получим правильное значение амплитуды сигнала.

По мере повышения громкости сигнала среднее этих значений будет увеличиваться, но некоторые из них являются отрицательными (в тех местах сигнала, где он

опускается ниже смещения постоянной составляющей), в результате чего они будут аннулировать друг друга и их среднее значение будет стремиться к нулю. Это проблема решается возведением каждого значения в квадрат (умножением на само себя). Это сделает все значения положительными, а также увеличит разницу между небольшими изменениями сигнала, что будет способствовать лучшей оценке таких изменений. Теперь среднее значение показаний будет увеличиваться и уменьшаться пропорционально амплитуде сигнала.

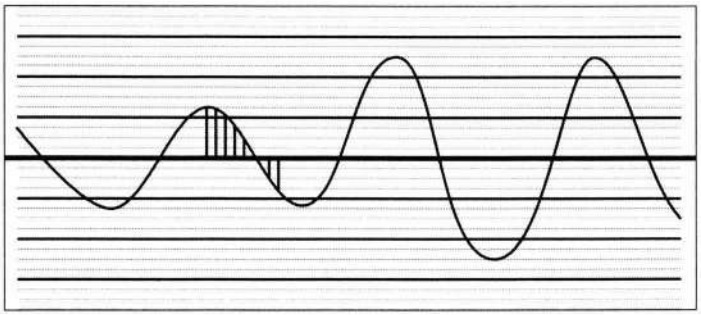


Рис. 6.11. Смещение постоянной составляющей звукового сигнала (средняя точка сигнала)

Для вычислений нужно знать, какое значение использовать для смещения постоянной составляющей. Чтобы получить чистый сигнал, усилитель микрофона должен обеспечивать смещение постоянной составляющей как можно ближе к середине возможного диапазона напряжений, чтобы мы получили сигнал как можно большей амплитуды без внесения в него искажений. В коде скетча предполагается это свойство и используется значение 512 (находящееся посередине диапазона значений входного диапазона от 0 до 1023). Каждый раз, когда скетч вычисляет среднее квадратов значений для получения нового показания, скетч обновляет текущее среднее значение. Текущее среднее вычисляется умножением текущего значения на значение:

```
(averagedOver - 1)
```

Значение переменной `averagedOver`, равное 16, «утяжеляет» текущее среднее в 15 раз. Затем скетч добавляет новое показание `newReading` (с весом, равным 1) и делит на значение `averagedOver`, получая средневзвешенное значение, которое становится новым текущим средним:

```
(currentAverage * 15 + newReading)/16
```

Скетч передает по последовательному интерфейсу значения новых показаний и текущего среднего, отформатированные таким образом, чтобы их можно было отображать в окне плоттера по последовательному соединению среды Arduino IDE (**Инструменты | Плоттер по последовательному соединению**). На рис. 6.12 показано взаимоотношение между новыми показаниями и текущими средними. График значений текущих средних более плавный, без всплесков, и это означает, что светодиод будет оставаться во включенном состоянии достаточно длительное время, чтобы быть замеченным, а не просто мигать кратковременно при всплеске.

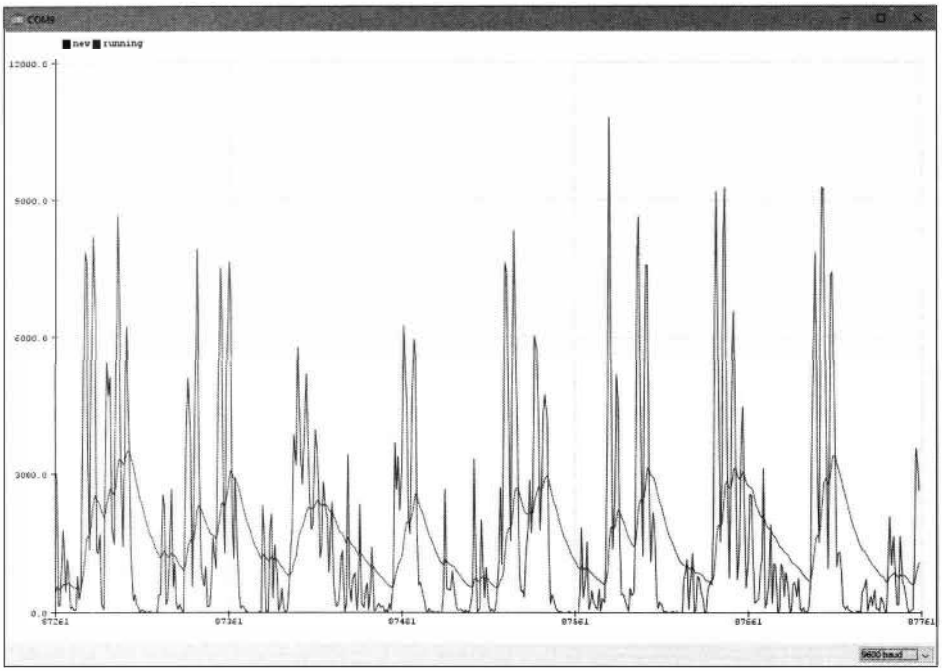


Рис. 6.12. Отображение значений показаний и текущего среднего в окне плоттера по последовательному соединению

Значения констант в начале скетча можно откорректировать, если светодиод не включается при требуемом уровне звука.

Например, для константы `numberOfSamples` задано значение 128. Если значение этой константы слишком малое, то текущее среднее значение может неудовлетворительно охватывать полные циклы сигнала, вследствие чего получаемые показания будут ошибочными. А при слишком большом значении этой константы текущее среднее будет вычисляться для слишком большого периода, в результате чего могут пропускаться очень короткие звуки, если они не создают достаточного изменения после усреднения значительного количества показаний. Слишком большое значение также может вносить заметную задержку между звуковым событием и соответствующим включением светодиода. Используемым в вычислениях константам, например `numberOfSamles` и `averagedOver`, присвоены значения, являющиеся степенью числа 2 (128 и 16 соответственно). Попробуйте добиться наилучшего быстродействия скетча, используя для этих констант значения, делимые на два без остатка (дополнительная информация по математическим функциям приведена в *главе 3*).

Если вычисляемые значения хорошо справляются с задачей обнаружения звуковых уровней, скетч можно модифицировать, чтобы использовать в нем стандартные методы для измерения уровня звука в децибелах. Для этого сначала нужно изменить способ вычисления нового показания `newReading` и вычислять квадратный корень из среднего (это называется *среднеквадратичным значением*). Затем надо взять десятичный логарифм обоих значений и умножить его на 20, чтобы получить децибелы.

Этот метод вряд ли даст точные показания без калибровки, но в целом вполне пригоден. Соответствующий фрагмент кода приводится в листинге 6.13.

Листинг 6.13. Вычисления уровня звука в децибелах

```
newReading = sqrt(sumOfSquares/numberOfSamples);
// Вычисляем скользящее среднее
runningAverage=(((averagedOver-1)*runningAverage)+newReading)/averagedOver;
Serial.print("new:"); Serial.print(20*log10(newReading));
Serial.print(",");
Serial.print("running:"); Serial.println(20*log10(runningAverage));
```

При этом также следует установить более низкое пороговое значение уровня звука для включения светодиода:

```
const int threshold = 30; // Пороговый уровень звука для включения светодиода
```

6.9. Измерение температуры

ЗАДАЧА

Требуется измерить температуру окружающей среды и вывести полученное значение на экран или использовать его для управления устройством — например, обогревателем или кондиционером.

РЕШЕНИЕ

Получить значение температуры окружающей среды можно с помощью датчика температуры TMP36. Этот датчик внешне выглядит как транзистор и подключается к плате Arduino, как показано на рис. 6.13. Скетч для работы с датчиком приводится в листинге 6.14.

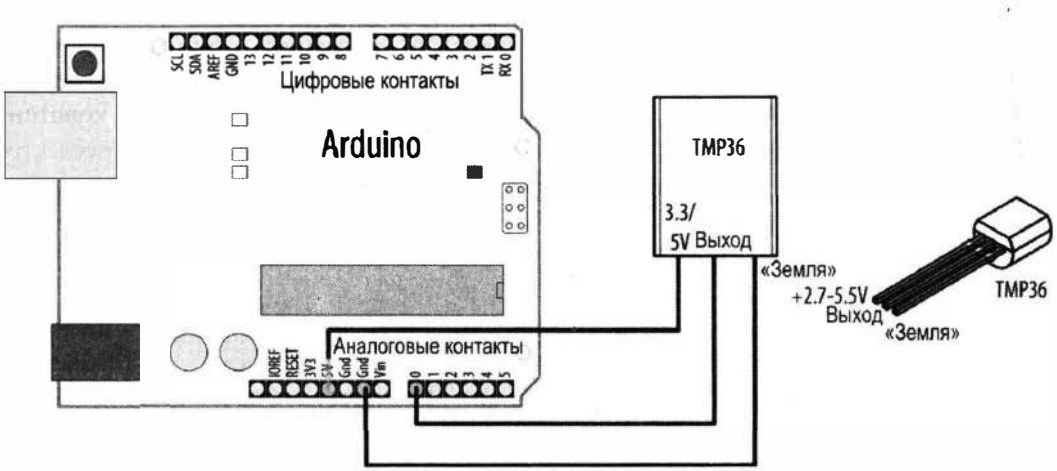


Рис. 6.13. Подключение датчика температуры TMP36 к плате Arduino

При использовании платы с напряжением питания 3,3 В вывод питания датчика необходимо подключить не к контакту 5V платы Arduino, а к контакту 3V3. Кроме того, в строке кода скетча:

```
float millivolts = (value / 1024.0) * 5000;
```

число 5000 нужно заменить числом 3300:

```
float millivolts = (value / 1024.0) * 3300;
```

Листинг 6.14. Скетч для работы с датчиком температуры TMP36

```
/*
* Скетч tmp36
* Измеряет температуру окружающей среды и отображает значение в окне монитора порта
* При достижении определенного порогового значения температуры
  включает встроенный светодиод
*/

const int inPin = A0;          // Номер контакта платы Arduino
                               // для подключения контакта выходного сигнала датчика
const int ledPin = LED_BUILTIN;
const int threshold = 80;      // Пороговое значение температуры
                               // 80°F (26,7°C) для включения светодиода

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

void loop()
{
   int value = analogRead(inPin);

   // Для плат с питанием 3,3 В замените 5000 на 3300
   float millivolts = (value / 1024.0) * 5000;
   // 10 мВ на градус Цельсия со смещением в 500 мВ
   float celsius = (millivolts - 500) / 10;
   float fahrenheit = (celsius * 9)/ 5 + 32;

   Serial.print("C:");
   Serial.print(celsius);
   Serial.print(",");
   Serial.print("F:");
   Serial.println( fahrenheit ); // Преобразовывает в градусы по Фаренгейту

   if (fahrenheit > threshold)
   {
      // Пороговое значение превышено?
      digitalWrite(ledPin, HIGH); // Если да, включаем светодиод
   }
```

```
    else
    {
      digitalWrite(ledPin, LOW); // Если нет, выключаем светодиод
    }
    delay(1000); // Ждем одну секунду
}
```

Обсуждение работы решения и возможных проблем

Датчик температуры TMP36 выдает аналоговый сигнал, напряжение которого прямо пропорционально измеренной температуре, с отношением 1 мВ на 0,1°С (или 10 мВ на градус), но со смещением величиной 500 мВ.

Скетч преобразовывает возвращаемые функцией `analogRead()` цифровые значения в милливольты (работа с функцией `analogRead()` подробно рассматривается в *главе 5*). Затем из полученного результата вычитается величина смещения 0,5 В (500 мВ) и результат делится на 10, чтобы получить показания в градусах по Цельсию. Если полученное показание температуры превышает заданное пороговое значение, включается встроенный светодиод платы. Датчик можно легко нагреть до температуры выше порогового значения 26,7°С, просто удерживая его между пальцами. При этом следует избегать прикосновения пальцами к выводам датчика, чтобы не вызвать искажений в показаниях.

На рынке доступно много разных типов датчиков температуры, и интересной альтернативой датчику TMP36 может стать датчик температуры DS18B20 в водонепроницаемом корпусе и с цифровым выходным сигналом. Он предлагается компанией Adafruit (артикул 381), компанией SparkFun (артикул SEN-11050) и другими поставщиками. Этот датчик подключается и используется иначе, чем датчик TMP36.

Выходной сигнал датчика DS18B20 передается по протоколу 1-Wire, впервые разработанному фирмой Dallas Semiconductor (сейчас — Maxim), и для работы с ним нужно установить две дополнительные библиотеки. Первая их них — Onewire. Список библиотек содержит несколько библиотек со словом «onewire» в названии, но нам нужна библиотека **Onewire** by Jim Studt, Tom Pollard... Вторая библиотека — Dallas Temperature (установка библиотек с помощью Менеджера библиотек среды Arduino IDE рассматривается в *разд. 16.2*). Подключение датчика DS18B20 к плате Arduino показано на рис. 6.14. В частности, красный провод датчика подключается к контакту питания 5V (или 3V3 в случае платы с напряжением питания 3,3 В) платы Arduino, черный провод — к контакту «земли» (Gnd), а провод выходного сигнала (желтого, белого или другого цвета) — к цифровому контакту 2. Выводы сигнала и «земли» датчика соединяются между собой резистором номиналом 4,7 кОм.

В листинге 6.15 приводится код скетча для работы с датчиком температуры DS18B20.

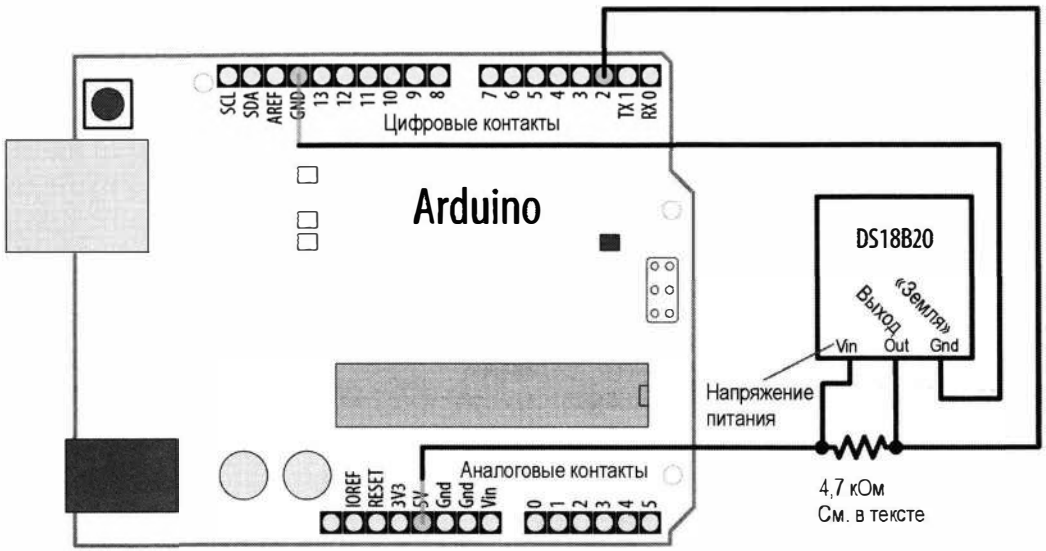


Рис. 6.14. Подключение датчика температуры DS18B20 к плате Arduino

Листинг 6.15. Скетч для работы с датчиком температуры DS18B20

```
/* Скетч DS18B20 temperature
* Считывает показания датчика температуры DS18B20 по протоколу 1-Wire
*/

#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 2 // Номер контакта, к которому подключается сигнал с датчика

const int ledPin = LED_BUILTIN;
const int threshold = 80; // Пороговое значение температуры
                          // 80°F (26,7°C) для включения светодиода

OneWire oneWire(ONE_WIRE_BUS);        // Подготавливаем подключение
                                      // по интерфейсу OneWire
DallasTemperature sensors(&oneWire); // Создаем временный экземпляр объекта датчика

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

   // Инициализируем датчик
   sensors.begin();
}
```

```
void loop(void)
{

   sensors.requestTemperatures(); // Запрашиваем показания температуры

   // Получаем показания температуры в градусах по шкалам
   // Фаренгейта и Цельсия
   float fahrenheit = sensors.getTempFByIndex(0);
   float celsius = sensors.getTempCByIndex(0);

   // Выводим показания температуры в формате для отображения
   // в плоттере по последовательному соединению
   Serial.print("C:"); Serial.print(celsius);
   Serial.print(",");
   Serial.print("F:"); Serial.println(fahrenheit);

   if (fahrenheit > threshold)
   {
      // Пороговое значение температуры превышено?
      digitalWrite(ledPin, HIGH); // Если да, включаем светодиод
   }
   else
   {
      digitalWrite(ledPin, LOW); // Если нет, выключаем светодиод
   }
   delay(1000);
}
```

В начале скетча подключаются заголовочные файлы библиотек и инициализируются структуры данных, необходимые для работы с протоколом 1-Wire и датчиком. В главном цикле скетча `loop()` датчику направляется запрос показания температуры, а затем считываются предоставленные показания в градусах по шкалам Цельсия и Фаренгейта. Обратите внимание на отсутствие необходимости выполнения преобразования показаний из одной шкалы в другую — показания предоставляются библиотекой в обеих шкалах. Также нет надобности в модифицировании кода при использовании платы с питанием 3,3 В. Просто в таком случае необходимо подключить провод питания датчика к контакту 3V3, а не к контакту 5V платы Arduino.

Дополнительная информация

Подробная информация по микросхеме датчика температуры TMP36 приводится в его справочном листке (**https://oreil.ly/gebtM**).

Подробная информация по микросхеме датчика температуры DS18B20 приводится в его справочном листке (**https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf**).

6.10. Чтение меток RFID (NFC)

ЗАДАЧА

Требуется считывать метки RFID/NFC и реагировать на определенный номер ID.

РЕШЕНИЕ

Для считывания меток RFID/NFC можно использовать считыватель на основе модуля PN532. Такие считыватели предлагаются несколькими поставщиками — вы можете взять, например, считыватель Grove-NFC компании Seeed Studio (артикул 113020006). Считыватели на основе модуля PN532 также предлагаются в виде шилдов компаниями SeeedStudio (артикул 113030001) и Adafruit (артикул 789). Считыватель PN532 подключается к плате Arduino через контакты последовательного интерфейса (TX и RX), как показано на рис. 6.15. Для работы со считывателем надо установить библиотеку Seeed_Arduino_NFC (**https://oreil.ly/JwEpb**). Подробная информация по установке библиотек излагается в *разд. 16.2*. Библиотека Seeed включает модифицированную версию библиотеки NDEF (**https://oreil.ly/Aepnl**), поэтому специально устанавливать эту библиотеку не нужно.

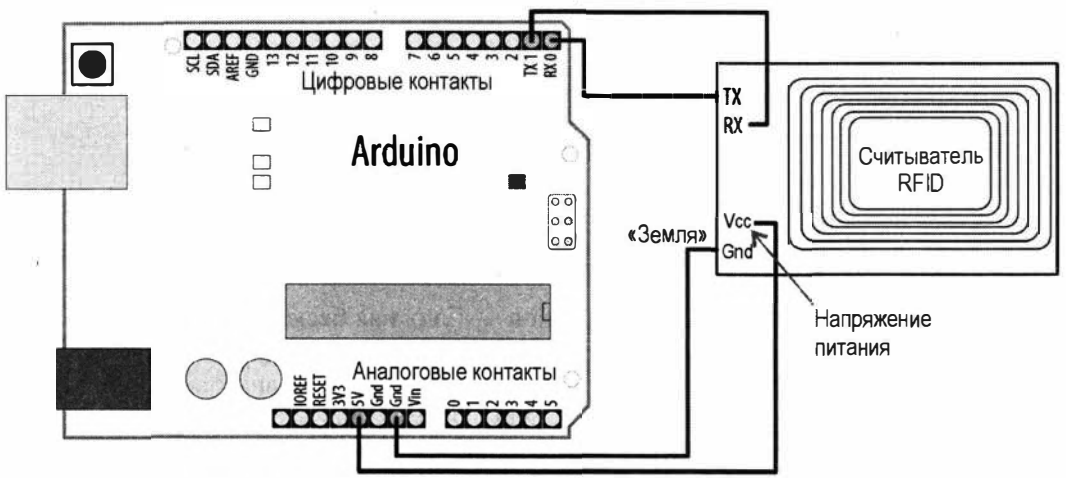


Рис. 6.15. Подключение считывателя RFID/NFC PN532 к плате Arduino

Считыватели PN532 работают в частотном диапазоне 13,56 МГц с метками MIFARE Classic и MIFARE Ultralight. Если вы используете другой считыватель, обратитесь к его документации за информацией по его подключению к плате Arduino и примерам кода.

Приведенный в листинге 6.16 скетч считывает с помощью считывателя PN532 содержимое метки NFC и отображает ее уникальный номер ID в окне монитора порта.

Листинг 6.16. Скетч для считывания меток NFC

```
/* Скетч NFC Tag Scanner - Serial
* Считывает метку NFC и отображает ее ID-номер в окне монитора порта
*/

#include <NfcAdapter.h>
#include <PN532/PN532/PN532.h>
#include <PN532/PN532_HSU/PN532_HSU.h>

PN532_HSU pn532hsu(Serial1);
NfcAdapter nfc(pn532hsu);

void setup()
{
   Serial.begin(9600);
   nfc.begin(); // Инициализируем считывание NFC
}

void loop()
{
   Serial.println("Waiting for a tag"); // Ожидаем метку
   if (nfc.tagPresent()) // Если считыватель видит NFC-метку
   {
      NfcTag tag = nfc.read();            // считывает ее содержимое
      Serial.println(tag.getUidString()); // и отображает ее ID-номер
   }
   delay(500);
}
```

Обсуждение работы решения и возможных проблем

Технология NFC (Near Field Communication — ближняя бесконтактная связь) — это специализированная версия технологии RFID (Radio Frequency Identification — радиочастотная идентификация), работающая в частотном диапазоне 13,56 МГц и поддерживающая формат данных NDEF (NFC Data Exchange Format — NFC-формат обмена данными). Формат обмена данными NDEF определяет набор структурированных сообщений, которые можно хранить на метке. *Метка* (или *тег*, от *англ.* tag) представляет собой небольшое электронное устройство, которое можно встраивать в визитные карточки, наклейки, брелоки и прочие подобные объекты. Метка содержит сравнительно большую антенну, которая принимает сигналы от считывателя RFID/NFC. Считыватель может быть встроен в компьютер или мобильный телефон, а также выполнен в виде отдельного модуля (как считыватель PN532), который можно подключать к другим устройствам — например, к плате Arduino. Когда метка принимает сигнал от считывателя, она при этом получает от него достаточно энергии для питания своей электронной схемы, которая отвечает на сигнал, передавая содержащуюся в памяти метки информацию. Существуют также метки, обладающие встроенным источником питания, — например, транспондеры, устанавли-

ваемые в автомобилях для автоматической оплаты за проезд. Такие метки называются *активными*, тогда как метки, питаемые опрашивающим сигналом, называются *пассивными*.

При активировании сигналом считывателя метка NDEF передает ему набор данных, содержащий информацию, идентифицирующую метку, а также хранящуюся на метке информацию. В скетче из листинга 6.17 используется библиотека NDEF Дона Коулмана (Don Coleman), упрощающая считывание данных с метки.

Код решения будет работать и со считывателем Grove-NFC компании Seeed Studio, подключенным к последовательному порту `Serial1`. Для передачи считываемой с метки информации в окно монитора порта скетч задействует последовательный порт USB-Serial. Плата Arduino Uno оснащена только одним аппаратным последовательным портом (см. *разд. «Аппаратные средства для последовательной связи» главы 4*), который занят организацией USB-Serial подключения к компьютеру. Поэтому для взаимодействия со считывателем NFC в решении используется программный последовательный порт, создаваемый с помощью библиотеки SoftwareSerial (работа с программным последовательным портом подробно рассматривается в *разд. 4.11*).

Чтобы использовать скетч решения с шилдом считывателя Grove-NFC компании Seeed Studio в режиме SPI, замените строки кода в начале скетча решения строками из листинга 6.17.

Листинг 6.17. Настройки для работы с модулем Grove-NFC по протоколу SPI

```
#include <SPI.h>
#include <NfcAdapter.h>
#include <PN532/PN532/PN532.h>
#include <PN532/PN532_SPI/PN532_SPI.h>

PN532_SPI pn532spi(SPI, 10);
NfcAdapter nfc = NfcAdapter(pn532spi);
```

Модуль Grove-NFC можно также сконфигурировать для взаимодействия через интерфейс I^2C (**https://oreil.ly/76Noz**). И чтобы использовать скетч решения со считывателем Grove-NFC компании Seeed Studio или считывателем компании Adafruit в этом режиме, замените строки кода в начале скетча решения строками из листинга 6.18.

Листинг 6.18. Настройки для работы со считывателем PN532 по протоколу I^2C

```
#include <Wire.h>
#include <NfcAdapter.h>
#include <PN532/PN532/PN532.h>
#include <PN532/PN532_I2C/PN532_I2C.h>

PN532_I2C pn532i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532i2c);
```

Кроме считывания информации с метки библиотека NDEF позволяет записывать информацию на метку (при условии, что метка не заблокирована для записи). Если заменить функцию `loop()` решения версией кода из листинга 6.19, скетч выполнит считывание метки и через функцию `print()` объекта `NfcTag` отобразит ID-номер метки и прочую считанную информацию в окне монитора порта. Затем в окне монитора порта будет выводиться обратный отсчет. Если метку оставить на месте до завершения отсчета, на нее будет записан предусмотренный в коде интернет-адрес (URL). Если теперь эту метку поднести к смартфону, поддерживающему возможности NFC, то в браузере должна открыться веб-страница, расположенная по этому адресу.

Листинг 6.19. Функция `loop()` для записи информации на метку

```
void loop()
{
   Serial.println("Waiting for a tag"); // Ожидание метки
   if (nfc.tagPresent()) // Если считыватель видит NFC-метку
   {
      NfcTag tag = nfc.read(); // считывает ее содержимое
      tag.print(); // выводит в окно монитора порта содержимое метки

      // Предоставляет пользователю возможность убрать метку,
      // чтобы не записывать на нее
      Serial.print("Countdown to writing the tag: 3");
                  // Обратный отсчет до записи на метку
      for (int i = 2; i >= 0; i--)
      {
         delay(1000);
         Serial.print("..."); Serial.print(i);
      }
      Serial.println();

      // Write a message to the tag
      NdefMessage message = NdefMessage();
      message.addUriRecord("http://oreilly.com");
      bool success = nfc.write(message);
      if (!success)
         Serial.println("Write failed."); // Не удалось выполнить запись на метку
      else
         Serial.println("Success.");   // Запись выполнена успешно
   }
      delay(500);
}
```

6.11. Отслеживание вращательного движения

ЗАДАЧА

Требуется определять вращение объекта, чтобы отслеживать его скорость и/или направление.

РЕШЕНИЕ

Эта задача решается посредством поворотного энкодера, который закрепляется на объекте, вращение которого нужно отслеживать. Подключение поворотного энкодера к плате Arduino показано на рис. 6.16, а в листинге 6.20 приводится скетч для работы с ним.

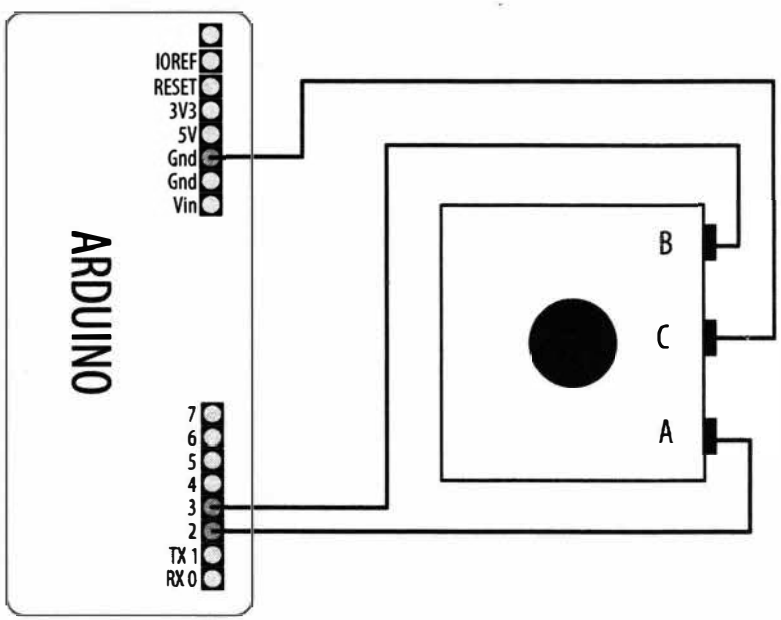


Рис. 6.16. Подключение поворотного энкодера к плате Arduino

Листинг 6.20. Скетч для работы с поворотным энкодером

```
/*
* Скетч Read a rotary encoder
* Скетч опрашивает контакты выходного сигнала энкодера
* Полученная информация о положении отображается в окне монитора порта

  */
  const int encoderPinA = 3;
  const int encoderPinB = 2;
  const int encoderStepsPerRevolution=16;
```

```
int angle = 0;
int encoderPos = 0;

bool encoderALast = LOW; // Булева переменная для хранения предыдущего
                         // состояния контакта

void setup()
{
   Serial.begin (9600);
   pinMode(encoderPinA, INPUT_PULLUP);
   pinMode(encoderPinB, INPUT_PULLUP);
}

void loop()
{
   bool encoderA = digitalRead(encoderPinA);

   if ((encoderALast == HIGH) && (encoderA == LOW))
   {
      if (digitalRead(encoderPinB) == LOW)
      {
         encoderPos--;
      }
      else
      {
         encoderPos++;
      }

      angle=(encoderPos % encoderStepsPerRevolution)*360/encoderStepsPerRevolution;
      Serial.print (encoderPos);
      Serial.print (" ");
      Serial.println (angle);
   }

   encoderALast = encoderA;
}
```

Обсуждение работы решения и возможных проблем

В ответ на вращение вала поворотный энкодер генерирует два сигнала. В процессе вращения вала уровни обоих этих сигналов находятся в диапазоне между высоким (`HIGH`) и низким (`LOW`), но при этом сигналы слегка не совпадают по фазе друг с другом. В точке перехода одного сигнала с высокого уровня на низкий состояние другого сигнала (высокий уровень или низкий) указывает на направление вращения вала энкодера.

В первой строке кода функции `loop()` выполняется чтение состояния контакта, к которому подключен один из выходных сигналов энкодера:

```
int encoderA = digitalRead(encoderPinA);
```

Затем полученное состояние контакта сравнивается с его предыдущим состоянием, чтобы определить, не поменялось ли оно только что с высокого на низкий:

```
if ((encoderALast == HIGH) && (encoderA == LOW))
```

Если состояние не поменялось, следующий блок кода не исполняется, а управление переходит в конец функции `loop()`, где только что считанное значение сохраняется в переменной `encoderALast`, после чего цикл повторяется.

Если же состояние поменялось, то код считывает состояние контакта, к которому подключен другой выходной сигнал кодера, и инкрементирует или декрементирует значение переменной `encoderPos` в зависимости от возвращенного значения. Затем вычисляется угол проворота вала энкодера (при этом за 0° принимается точка, в которой вал энкодера находился, когда код начал исполняться). Полученные значения отображаются в окне монитора порта.

Одной из основных характеристик поворотных энкодеров является *разрешение*, определяемое как количество шагов на оборот, т. е. количество изменений сигналов кодера между высоким и низким уровнем за один поворот вала. На рынке предлагаются энкодеры с разрешением в диапазоне от 16 до 1000. Энкодеры с большим разрешением могут обнаруживать малые перемещения, но они также и более дорогие. Разрешение для энкодера, задействованного в нашем решении, прописано в коде скетча:

```
const int encoderStepsPerRevolution=16;
```

В случае использования энкодера с другим разрешением эту строку кода нужно модифицировать, указав его разрешение, чтобы получить правильные угловые значения.

Если при повороте вала энкодера значения только увеличиваются, независимо от направления вращения, попробуйте выявлять нарастающий фронт сигнала, а не спадающий. Для этого в строке кода соответствующей проверки значения `LOW` и `HIGH` нужно поменять местами:

```
if ((encoderALast == LOW) && (encoderA == HIGH))
```

Поворотные энкодеры выдают только сигнал инкрементирования или декрементирования, но не могут предоставлять прямую информацию об угле проворота своего вала. Этот угол вычисляется кодом, но рассчитывается относительно начальной позиции при каждом запуске кода. Код отслеживает состояние контактов, на которые подаются сигналы энкодера, методом опроса. Такой подход не гарантирует обнаружение изменения состояния между двумя последовательными опросами. Поэтому, если кроме обслуживания энкодера код выполняет какую-либо дополнительную работу, а также при быстром вращении вала энкодера, существует возможность, что некоторые шаги будут пропущены. Такая ситуация более вероятна для энкодеров с высоким разрешением, поскольку при повороте вала они посылают сигналы с более высокой частотой.

Чтобы вычислить скорость вращения вала энкодера, нужно посчитать количество зарегистрированных шагов в определенном направлении за заданный период времени.

6.12. Отслеживание вращения вала кодера с использованием прерываний

ЗАДАЧА

Если кроме обслуживания энкодера код выполняет дополнительные задачи, или если требуется обслуживать несколько энкодеров, код может не успевать регистрировать все шаги поворота вала между двумя последовательными опросами, что приведет к возвращению неправильных показаний о положении вала энкодера. Эта проблема особенно усугубляется быстрым вращением вала энкодера. Требуется ее решить.

РЕШЕНИЕ

Эта задача — быстрого реагирования на изменения состояния контактов входных сигналов — решается путем использования возможностей платы Arduino по обнаружению *прерываний* (тема прерываний подробно рассматривается в *разд. 18.2*). Для этого нам потребуется установить библиотеку Encoder от Поля Стоффрегена (Paul Stoffregen), которая оптимизирована для работы с поворотными энкодерами (установка библиотек подробно рассматривается в *разд. 16.2*). Энкодер подключается так же, как в предыдущем решении. Использующий прерывания скетч для работы с энкодером приводится в листинге 6.21.

Листинг 6.21. Обслуживание поворотного энкодера с использованием прерываний

```
/* Скетч Rotary Encoder library
* Отслеживает состояние сигналов поворотного энкодера, используя прерывания
*/

#include <Encoder.h>

Encoder myEnc(2, 3); // Для плат MKR используются контакты 6, 7

void setup()
{
   Serial.begin(9600);
}
long lastPosition = -999;

void loop()
{
   long currentPosition = myEnc.read();
   if (currentPosition != lastPosition)
   {
      // If the position changed
      lastPosition = currentPosition;   // Сохраняем последнее положение
```

```
        Serial.println(currentPosition);  // Отображаем значение в окне монитора порта
    }
}
```

Обсуждение работы решения и возможных проблем

Когда коду в решении из *разд. 6.11* (см. листинг 6.21) кроме обслуживания кодера нужно выполнять другие задачи, он будет реже опрашивать контакты сигналов энкодера. И если состояние одного из сигналов изменится, прежде чем код сможет опросить соответствующий контакт, этот шаг просто не будет обнаружен. Быстрое вращение вала энкодера также может вызывать дополнительные ошибки, поскольку изменения состояний (шаги) будут происходить более часто.

Чтобы обеспечить обнаружение каждого шага, необходимо использовать прерывания. Когда происходит событие прерывания (в нашем случае — изменение состояния контакта), код скетча бросает все, чем он занимается в текущий момент, и переходит к обработке прерывания. Завершив обработку прерывания, код возвращается к точке, где произошло прерывание, и продолжает исполнять прерванную задачу. Библиотека Encoder лучше всего работает с контактами, которые поддерживают аппаратные прерывания, но будет прилагать максимум усилий и при работе с контактами, не поддерживающими прерываний.

На плате Arduino Uno и других платах с микроконтроллером ATmega328 для прерываний можно использовать только два контакта: 2 и 3. Список контактов, поддерживающих аппаратные прерывания, для других плат приводится на веб-странице **https://oreil.ly/jUYkM** сайта Arduino. Объект поворотного энкодера объявляется и инициализируется следующим кодом:

```
Encoder myEnc(2, 3);
```

В параметрах инициализации объекта `Encoder` указываются два контакта платы Arduino, на которые подаются выходные сигналы кодера. В случае, если возвращаемые значения кодера уменьшаются, когда ожидается их увеличение, поменяйте местами эти параметры или физические подключения выходных сигналов кодера к контактам платы Arduino. После инициализации объекта `Encoder` вращение вала энкодера будет вызывать прерывание исполнения текущей задачи скетча для обработки отслеживания перемещения вала энкодера. Чтение состояний сигналов кодера осуществляется функцией `myEnc.read()`.

Можно создать любое количество объектов энкодера, для которого хватит контактов платы Arduino, но по мере возможности следует использовать контакты, поддерживающие аппаратные прерывания. В листинге 6.22 приводится скетч, который может одновременно обслуживать два поворотных энкодера. Он будет оптимально работать с платами, поддерживающими аппаратные прерывания на указанных контактах, — например, с платами, оснащенными микроконтроллером SAMD21 (плата Metro M0 компании Adafruit, плата RedBoard Turbo компании SparkFun и плата Arduino Zero). При использовании других плат может потребоваться указать иные номера контактов для подключения энкодеров. Плата Arduino Uno и другие платы

с микроконтроллером ATmega328 поддерживают аппаратные прерывания только на контактах 2 и 3, поэтому для таких плат качество показаний для второго поворотного энкодера ухудшится, независимо от того, к каким контактам он будет подключен.

Листинг 6.22. Использование прерываний для обслуживания двух поворотных энкодеров

```
#include <Encoder.h>

Encoder myEncA(2, 3);  // Для плат MKR используются контакты 4, 5
Encoder myEncB(6, 7);  // Для плат Mega используются контакты 18, 19

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

long lastA = -999;
long lastB = -999;

void loop()
{
   long currentA = myEncA.read();
   long currentB = myEncB.read();
   if (currentA != lastA || currentB != lastB)
   {
      // Если положение любого кодера изменилось
      lastA = currentA; // Сохраняем оба положения
      lastB = currentB;

      // Отображаем показания в окне монитора порта
      // (или плоттера по последовательному соединению)
      Serial.print("A:"); Serial.print(currentA);
      Serial.print(" ");
      Serial.print("B:"); Serial.println(currentB);
   }
}
```

Дополнительная информация

Плата Arduino MKR Vidor 4000 (**https://oreil.ly/KqIIr**) содержит вентильную матрицу FPGA, которая позволяет считывать поворотные энкодеры с намного большей точностью, чем это может делать обычная плата Arduino.

6.13. Работа с мышью

ЗАДАЧА

Требуется определять движения мыши типа PS/2 и реагировать на изменения координат X и Y.

РЕШЕНИЕ

Перемещения мыши мы будем обозначать с помощью светодиодов. В частности, яркость светодиодов станет меняться в ответ на перемещения мыши по оси X (влево и вправо) и по оси Y (дальше и ближе). Щелчок кнопкой мыши задает ее текущее положение как контрольную точку. Подключение светодиодов и мыши к плате Arduino показано на рис. 6.17.

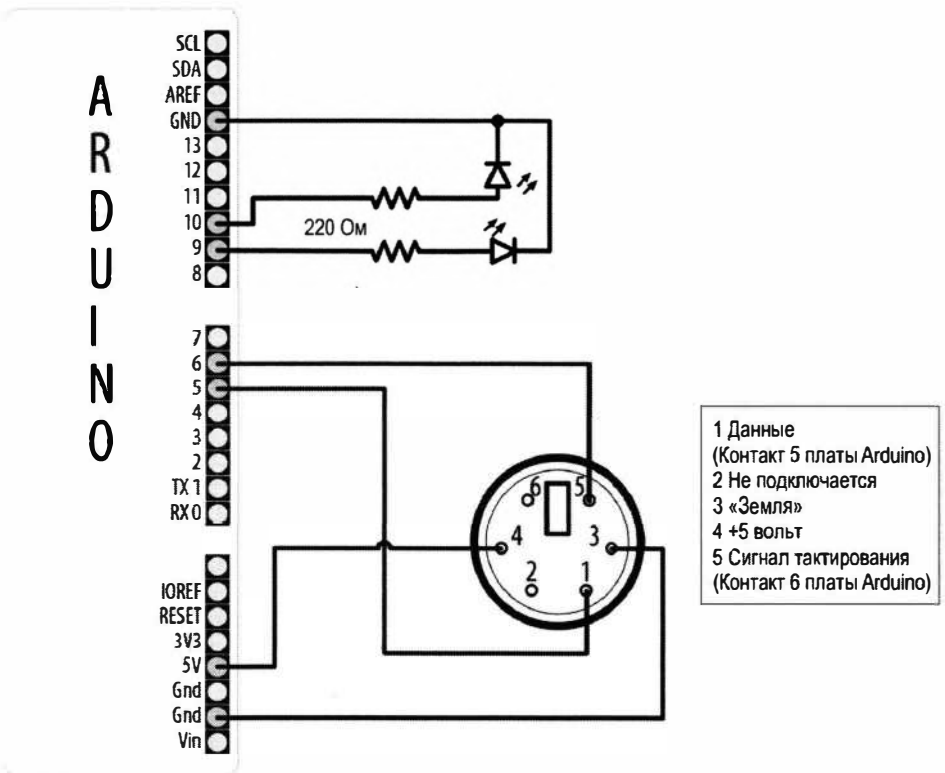


Рис. 6.17. Подключение мыши PS/2 и светодиодов к плате Arduino

При использовании платы с напряжением питания 3,3 В сигналы тактирования и данных нужно подавать на контакты платы Arduino через делитель напряжения. Впрочем, на мышь можно попробовать подавать питание 3,3 В вместо 5 В (это может сработать или нет, в зависимости от используемой мыши). Более подробная информация по работе с делителем напряжения приведена в *разд. 5.11*.

На рис. 6.17 гнездовой разъем PS/2 (в который вставляется штыревой разъем мыши) показан спереди. При отсутствии такого гнездового разъема можно обрезать разъем мыши (если вы можете себе это позволить) и припаять провода к штыревым разъемам, которые потом можно вставлять напрямую в гнездовые разъемы платы Arduino. При этом нужно запомнить соответствие проводов контактам разъема. Определить соответствие проводов контактам гнездового разъема мыши можно с помощью простой прозвонки (проверки на проводимость) между контактами и проводами. Также следует иметь в виду, что соответствие проводов контактам штыревого разъема мыши будет зеркальным по вертикали относительно цоколевки, показанной на рис. 6.17.

Скетч для работы с этой схемой приводится в листинге 6.23. Но для работы скетча нужно установить библиотеку PS/2 (**https://oreil.ly/NSl9T**). Кроме того, на момент подготовки этой книги нужно было подредактировать файл **ps2.h** библиотеки, расположенный в папке **ps2**. Для этого откройте этот файл в текстовом редакторе и замените код

```
#include "WProgram.h"
```

на

```
#include "Arduino.h".
```

Листинг 6.23. Скетч для роботы с мышью PS/2

```
/*
Скетч Mouse
Обрабатывает перемещения мыши PS/2, используя библиотеку ps2
с веб-сайта http://www.arduino.cc/playground/ComponentLib/Ps2mouse
*/

#include <ps2.h>

const int dataPin = 5;
const int clockPin = 6;

const int xLedPin = 9; // Для плат MKR используем контакт 8
const int yLedPin = 10;

const int mouseRange = 255; // Максимальный диапазон значений осей X/Y

char x;                     // Показания мыши
char y;
byte status;

int xPosition = 0;      // При перемещениях мыши значения инкрементируются
                        // и декрементируются
int yPosition = 0;
```

```
int xBrightness = 128;  // Значения увеличиваются и уменьшаются
                        // в зависимости от положения мыши
int yBrightness = 128;

const byte REQUEST_DATA = 0xeb; // Команда на получение данных с мыши

PS2 mouse(clockPin, dataPin);   // Объявляем экземпляр объекта мыши

void setup()
{
  mouseBegin(); // Инициализируем объект мыши
}

void loop()
{
  // Считываем данные с мыши
  mouse.write(REQUEST_DATA); // Запрашиваем данные с мыши
  mouse.read();              // Игнорируем подтверждение запроса

  status = mouse.read();     // Считываем состояния кнопок мыши
  if(status & 1) // Этот бит установлен, если нажата левая кнопка мыши
  xPosition = 0; // Центрируем положение мыши по оси X
  if(status & 2) // Этот бит установлен, если нажата правая кнопка мыши
  yPosition = 0; // Центрируем положение мыши по оси Y

  x = mouse.read();
  y = mouse.read();
  if( x != 0 || y != 0)
  {
    // Переходим сюда при наличии перемещений мыши

    xPosition = xPosition + x; // Накапливаем значения положения
    xPosition = constrain(xPosition,-mouseRange,mouseRange);

    xBrightness = map(xPosition, -mouseRange, mouseRange, 0,255);
    analogWrite(xLedPin, xBrightness);

    yPosition = constrain(yPosition + y, -mouseRange,mouseRange);
    yBrightness = map(yPosition, -mouseRange, mouseRange, 0,255);
    analogWrite(yLedPin, yBrightness);
  }
}

void mouseBegin()
{
  // Выполняем сброс и инициализацию мыши
  mouse.write(0xff); // Сброс
  delayMicroseconds(100);
```

```
    mouse.read(); // Байт подтверждения
    mouse.read(); // Бланк
    mouse.read(); // Бланк
    mouse.write(0xf0); // Удаленный режим
    mouse.read(); // Подтверждение
    delayMicroseconds(100);
}
```

Обсуждение работы решения и возможных проблем

Подключите мышь к плате Arduino, как показано ранее на рис. 6.17. Данное решение применимо только к устройствам типа PS/2, но не к устройствам, подключаемым через USB-разъем. Поэтому вам придется найти подходящую мышь с разъемом PS/2.

Функция `mouseBegin()` инициализирует объект мыши, чтобы он мог реагировать на перемещения физической мыши и нажатия ее кнопок. Низкоуровневая связь между мышью и платой обеспечивается библиотекой PS/2. Команда `mouse.write()` извещает мышь о том, что будут запрашиваться данные. Мышь подтверждает первый вызов функции `mouse.read()`, но это подтверждение в нашем примере игнорируется. Следующий вызов функции `mouse.read()` считывает состояние кнопок, а два последних ее вызова считывают значения перемещений по осям X и Y, которые были выполнены после предыдущего запроса.

Чтобы определить нажатую кнопку мыши (правая или левая), скетч проверяет, какие биты в значении переменной `status` имеют значение `HIGH`. Нажатие левой и правой кнопок устанавливают два самых правых бита этого значения, которые и проверяются кодом в листинге 6.24.

Листинг 6.24. Проверка битов на нажатие кнопок мыши

```
status = mouse.read(); // Считываем состояния кнопок мыши
if(status & 1) // Установлен самый правый бит, если нажата
               // левая кнопка мыши
   xPosition = 0; // Центрируем положение мыши по оси X
if(status & 2) // Установлен этот бит, если нажата правая кнопка мыши
   yPosition = 0; // Центрируем положение мыши по оси Y
```

Считываемые с мыши значения X и Y представляют ее перемещения по этим осям после предыдущего запроса. Эти значения накапливаются в переменных `xPosition` и `yPosition`.

Значения X и Y будут положительными при перемещении мыши вправо или от себя и отрицательными при перемещении влево и на себя.

Чтобы не допустить превышения границ заданного диапазона (`mouseRange`), накапливаемые значения проверяются с помощью функции `constrain()`:

```
xPosition = xPosition + x; // Накапливаем значения положения
xPosition = constrain(xPosition,-mouseRange,mouseRange);
```

Для вычисления положения по оси Y (yPosition) применяется сокращенная версия кода, где вычисление значения Y выполняется в вызове функции constrain():

```
yPosition = constrain(yPosition + y,-mouseRange,mouseRange);
```

При нажатии левой или правой кнопки мыши переменные xPosition и yPosition соответственно обнуляются.

Функция analogWrite() включает светодиоды с яркостью, соответствующей положению мыши: с половинной яркостью посередине диапазона перемещений и понижая или повышая яркость при увеличении или уменьшении значения положения мыши. Но для правильной работы этого подхода необходимо использовать контакт, поддерживающий работу с ШИМ-сигналом. Если плата не поддерживает ШИМ-сигналы на контактах 9 и 10 (большинство плат поддерживают), то вместо повышения и понижения яркости, светодиоды будут просто включаться и выключаться. Платы семейства MRK не поддерживают ШИМ-сигнал на контакте 9, поэтому для них необходимо модифицировать схему подключения и код, задействовав контакт, поддерживающий эту возможность.

Положение мыши можно отображать графически в плоттере по последовательному соединению, добавив следующую строку кода вызова функции printValues() сразу же после второго вызова функции analogWrite():

```
printValues(); // Отображаем значения X и Y в мониторе порта
               // или плоттере по последовательному соединению
```

Для этого также нужно добавить следующую строку кода в функцию setup():

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

Код функции printValues() для отображения значений положения мыши в мониторе порта или в плоттере по последовательному соединению приводится в листинге 6.25.

Листинг 6.25. Функция для отображения положения мыши в мониторе порта

```
void printValues()
{
   Serial.print("X:");
   Serial.print(xPosition);
   Serial.print(",Y:");
   Serial.print(yPosition);
   Serial.println();
}
```

Дополнительная информация

Гнездовой разъем PS/2 с припаянными проводами для подключения к плате Arduino можно приобрести в интернет-магазине компании Adafruit (**https://oreil.ly/eAiKx**).

6.14. Определение местонахождения с помощью системы GPS

ЗАДАЧА

Требуется определить местонахождение, используя модуль глобального позиционирования (GPS).

РЕШЕНИЕ

В настоящее время на рынке предлагается несколько модулей GPS, совместимых с платформой Arduino. Большинство этих модулей взаимодействуют с микроконтроллером платы Arduino посредством уже знакомого нам последовательного интерфейса по протоколу NMEA 0183. Этот стандартный отраслевой протокол обеспечивает доставку данных GPS принимающим устройствам в виде строк ASCII в удобном для понимания человеком формате. Например, следующая строка:

```
$GPGLL,4916.45,N,12311.12,W,225444,A,*1D
```

описывает, среди прочего, местоположение на земном шаре, имеющее координаты 49° 16.45' северной широты и 123° 11.12' западной долготы.

Чтобы определить местоположение, скетч Arduino должен выполнить парсинг таких строк и преобразовать соответствующий текст в цифровой формат. С учетом ограниченного адресного пространства платформы Arduino написание кода для извлечения данных из строк NMEA может оказаться сложной и обременительной задачей. Но, к счастью, эту задачу можно значительно облегчить с помощью библиотеки Микала Харта (Mikal Hart) TinyGPSPlus. Загрузите эту библиотеку со страницы GitHub Микала (**https://oreil.ly/jjnmf**) и установите ее (подробные инструкции по установке библиотек сторонних разработчиков представлены в *разд. 16.2*).

Общая процедура использования GPS следующая:

1. Физически подключаем GPS-устройство к плате Arduino.
2. Считываем с GPS-устройства последовательные данные в формате NMEA.
3. Обрабатываем считанные данные, чтобы определить местоположение.

Этот процесс реализуется с помощью библиотеки TinyGPSPlus следующим образом:

1. Физически подключаем GPS-устройство к плате Arduino.
2. Создаем экземпляр объекта `TinyGPSPlus`.
3. Считываем последовательные данные в формате NMEA с GPS-устройства.
4. Обрабатываем каждый байт считанных данных, используя метод (функцию) `encode()` объекта библиотеки TinyGPSPlus.
5. Периодически вызываем метод `get_position()` объекта библиотеки TinyGPSPlus, чтобы определить текущее местонахождение.

В листинге 6.26 приводится скетч, иллюстрирующий получение данных от GPS-модуля, подключенного к последовательному порту платы Arduino. Каждые пять секунд скетч мигает светодиодом один раз, если устройство находится в Южном полушарии, и два раза, если в Северном. Если контакты TX и RX вашей платы Arduino сопоставлены другому последовательному устройству — например, `Serial1`, откорректируйте определение `GPS_SERIAL` соответствующим образом (см. табл. 4.1).

Листинг 6.26. Получение данных о местонахождении с GPS-модуля

```
/* Скетч GPS
* Определяет земное полушарие местонахождения и соответственно
  мигает встроенным светодиодом
*/

#include "TinyGPS++.h"

// Укажите здесь последовательный порт, используемый вашим
// устройством GPS (Serial, Serial1, и т. д.)
#define GPS_SERIAL Serial

TinyGPSPlus gps; // Создаем экземпляр объекта TinyGPS++

#define HEMISPHERE_PIN LED_BUILTIN

void setup()
{
   GPS_SERIAL.begin(9600);    // Устройства GPS обычно
                              // взаимодействуют со скоростью 9600 бод
   pinMode(HEMISPHERE_PIN, OUTPUT);
   digitalWrite(HEMISPHERE_PIN, LOW); // В начале работы выключаем светодиоды
}

void loop()
{
   while (GPS_SERIAL.available())
   {
      // Обрабатываем функцией encode() каждый байт. Если функция
      // encode() возвращает значение true,
      // проверяем наличие данных нового местонахождения
      if (gps.encode(GPS_SERIAL.read()))
      {
         if (gps.location.isValid())
         {
            if (gps.location.lat() < 0) // Южное полушарие?
               blink(HEMISPHERE_PIN, 1);
```

```
            else
               blink(HEMISPHERE_PIN, 2);
         }
         else // Не удалось определить местоположение
            blink(HEMISPHERE_PIN, 5);
            delay(5000); // Пауза длительностью в 5 секунд
      }
   }
}

void blink(int pin, int count)
{
   for (int i = 0; i < count; i++)
   {
      digitalWrite(pin, HIGH);
      delay(250);
      digitalWrite(pin, LOW);
      delay(250);
   }
}
```

Последовательную связь запускаем на скорости, требуемой используемым устройством GPS. Тема последовательной связи для Arduino подробно рассматривается в *главе 4*.

В нашем случае последовательная связь с устройством GPS устанавливается на скорости 9600 бод. Поступающие по последовательному интерфейсу байты обрабатываются функцией `encode()`, которая выполняет парсинг NMEA-данных. Возвращение функцией `encode()` значения `true` означает успешное выполнение парсинга полной строки и возможное наличие данных для нового местонахождения. Здесь выполняется проверка действительности нового положения вызовом функции `gps.location.isValid()`.

Функция `gps.location.lat()` возвращает последние наблюдаемые данные широты, которые проверяются скетчем. Если результат меньше нуля (что означает широту к югу от экватора), выполняется двойное мигание светодиодом. Если же результат больше нуля (что означает широту к северу от экватора), светодиод мигает один раз. Если GPS-устройству не удалось определить местоположение, светодиод мигает пять раз.

Обсуждение работы решения и возможных проблем

Подключение GPS-модуля к плате Arduino — это предельно простая задача, и состоит она в подключении линий данных и питания модуля к соответствующим контактам платы Arduino, как показано в табл. 6.2. Для работы с платами, рассчитанными на питание 5 В (например, Arduino Uno), можно использовать модуль GPS с питанием как 3,3 В, так и 5 В. Но с платами, рассчитанными на питание 3,3 В (например, платами с микроконтроллером SAMD — такими как платы Arduino Zero,

Metro M0/M4 компании Adafruit или Redboard Turbo компании SparkFun), необходимо использовать GPS-модуль с питанием 3,3 В.

Таблица 6.2. *Подключение контактов GPS-модуля к контактам платы Arduino*

Контакт GPS-модуля	Контакт платы Arduino
GND	GND
5V или 3.3V	5V или 3V3
RX	TX (контакт 1)
TX	RX (контакт 0)

Некоторые GPS-модули выдают сигнал с уровнями RS-232, который несовместим с ТТЛ-логикой плат Arduino и безвозвратно повредит плату. Для подключения выходного сигнала таких GPS-модулей к плате Arduino необходимо использовать какое-либо устройство преобразования логических уровней — например, микросхему MAX232.

В коде решения предполагается, что GPS-устройство подключено непосредственно к контактам аппаратного последовательного порта платы Arduino. Но при использовании плат с микроконтроллером ATmega328 (например, платы Arduino Uno) — это не совсем удобный подход, т. к. линии RX и TX (контакты 0 и 1) их единственного аппаратного последовательного порта задействованы для интерфейса USB-Serial. И поскольку во многих проектах аппаратный последовательный порт платы занят для взаимодействия с локальным компьютером или другим периферийным устройством, этот порт нельзя использовать для взаимодействия с GPS-модулем. В таких случаях для подключения GPS-модуля следует воспользоваться программным последовательным портом, организованным на другой паре цифровых контактов платы.

Тем не менее в целях, например, отладки, вы можете освободить аппаратный последовательный порт платы Arduino, отключив питание от платы Arduino и GPS-модуля и переместив линию TX модуля GPS на контакт 2 платы Arduino, а линию RX — на контакт 3, как показано на рис. 4.8.

Сейчас же подключите плату Arduino к компьютеру, используя USB-кабель, и загрузите в нее скетч из листинга 6.27. Это даст возможность наблюдать показания GPS-модуля в окне монитора порта.

Листинг 6.27. Вывод показаний GPS-модуля в окно монитора порта

```
/* Скетч GPS with logging
*/

#include "TinyGPS++.h"

// Для плат, оснащенных дополнительным встроенным аппаратным
// последовательным портом, удалите или закомментируйте следующие
// четыре строки кода
```

```
#include "SoftwareSerial.h"
#define GPS_RX_PIN 2
#define GPS_TX_PIN 3
SoftwareSerial softserial(GPS_RX_PIN, GPS_TX_PIN); // Создаем объект
                                                   // программного последовательного порта

// Для плат, оснащенных дополнительным встроенным аппаратным
// последовательным портом, замените softserial
// идентификатором этого порта
#define GPS_SERIAL softserial

TinyGPSPlus gps; // Создаем экземпляр объекта TinyGPSPlus

void setup()
{
   Serial.begin(9600);     // Запускаем аппаратный последовательный порт
                        // для взаимодействия с компьютером (монитор порта)
   GPS_SERIAL.begin(9600); // Запускаем программный последовательный порт
                           // для взаимодействия с GPS-модулем
}

void loop()
{
   while (GPS_SERIAL.available())
   {
      int c = GPS_SERIAL.read();
      Serial.write(c); // Отображаем строку NMEA в окне монитора порта

      // Обрабатываем каждый полученный байт функцией encode()
      // Если функция encode() возвращает значение true,
      // проверяем наличие данных нового определения местонахождения
      if (gps.encode(c))
      {
         Serial.println();
         float lat = gps.location.lat();
         float lng = gps.location.lng();
         unsigned long fix_age = gps.date.age();

         if (!gps.location.isValid())
               Serial.println("Invalid fix"); // Недействительное определение
         else if (fix_age > 2000)
            Serial.println("Stale fix");      // Устарелое определение
         else
            Serial.println("Valid fix");      // Действительное определение

         Serial.print("Lat: ");
         Serial.print(lat);
```

```
            Serial.print(" Lon: ");
            Serial.println(lng);
        }
    }
}
```

Более подробная информация по работе с программным последовательным портом приводится в *разд. 4.11* и *4.12*.

Обратите внимание, что для аппаратного последовательного порта (взаимодействие с монитором порта) и программного последовательного порта (взаимодействие с GPS-модулем) можно задать разные скорости передачи данных.

Этот скетч (см. листинг 6.28) по большому счету работает так же, как и скетч из листинга 6.27 (не используя для краткости функциональность мигания светодиодом), но для него намного проще выполнять отладку. Теперь к встроенному аппаратному порту платы Arduino можно подключить ЖКД с последовательным интерфейсом (см. *разд. 4.11*), и на него будут выводиться строки NMEA с GPS-монитора и данные местонахождения, извлеченные из них скетчем. Впрочем, к встроенному аппаратному порту можно просто подключить монитор порта среды Arduino IDE, и эти данные будут выводиться в окно порта.

При включении питания GPS-устройство начинает передавать строки в формате NMEA. Но строки, содержащие действительные данные местонахождения, передаются только после того, как это устройство реально установит местонахождение. Для этого необходимо, чтобы антенна GPS-устройства имела беспрепятственный обзор неба для приема сигналов спутников системы GPS, а процесс определения местонахождения может занять до двух минут или даже больше. Грозовая погода или наличие зданий или других препятствий между антенной и спутниками также может отрицательно воздействовать на возможность GPS-модуля определять местонахождение. Тогда как же скетч узнает, являются ли действительными данные местонахождения, выдаваемые библиотекой TinyGPSPlus? По значению, возвращаемому функцией `gps.location.isValid()`. Значение `false` означает, что на текущий момент еще не был выполнен парсинг действительных строк, содержащих данные о местонахождении. Это дает нам знать, что возвращенные данные широты и долготы недействительны.

Можно также уточнить, сколько времени прошло после последней попытки определения местонахождения. Это делается с помощью функции `gps.date.age()`, которая возвращает количество миллисекунд, прошедших с последней попытки определения местонахождения. Возвращаемое этой функцией значение сохраняется в переменной `fix_age`. При нормальных условиях работы значение `fix_age` должно было достаточно низким. Современные устройства GPS способны возвращать данные о местонахождении от одного до пяти раз в секунду или даже чаще, поэтому значение `fix_age` свыше 2000 мс или около того может означать наличие проблем с определением местонахождения. Возможно, устройство GPS находится внутри туннеля, или данные NMEA искажаются неправильным подключением, создавая ошибку контрольной суммы (контрольная сумма вычисляется для проверки отсутствия ис-

кажений данных). В любом случае слишком большое значение переменной `fix_age` означает, что возвращенные функцией `get_position()` координаты устарели.

Дополнительная информация

Дополнительную информацию о протоколе NMEA можно получить из различных интернет-источников, в том числе и из статей в Википедии (**https://oreil.ly/APOzP**).

На рынке присутствует множество модулей GPS, способных взаимодействовать с платой Arduino и библиотекой TinyGPS. Эти модули отличаются друг от друга только энергопотреблением, напряжением питания, точностью, физическим интерфейсом и наличием или отсутствием поддержки передачи данных в формате NMEA по последовательному интерфейсу. Большой выбор модулей GPS предлагают такие компании, как Adafruit (**https://oreil.ly/9rN5V**) и SparkFun (**https://oreil.ly/w0asL**).

Технология GPS подвигла многих разработчиков на создание интересных проектов на основе платформы Arduino. Одним из чрезвычайно популярных таких проектов является регистратор данных GPS, в котором перемещающееся устройство записывает данные о местонахождении в память EEPROM[6] платы Arduino или в другой тип хранилища данных. С примером такого проекта можно познакомиться на веб-страницах: **https://oreil.ly/w0asL** и **https://oreil.ly/YBhAI**. Компания Adafruit предлагает популярный шилд регистрации данных GPS (**https://oreil.ly/8eOqy**).

Следует отметить и такие интересные GPS-проекты, как радиоуправляемые модели самолетов и вертолетов, которые следуют заданному маршруту от одной точки к другой под управлением команд скетча Arduino.

А Микал Харт создал «шкатулку сокровищ» на основе GPS, внутренняя защелка которой открывается только тогда, когда шкатулка физически находится в месте с определенными координатами. С описанием этого проекта можно ознакомиться на веб-странице **https://oreil.ly/FAvDD**.

6.15. Определение вращения с помощью гироскопа

ЗАДАЧА

Требуется реагировать на скорость вращения объекта. Эту возможность можно использовать, например, чтобы транспортное средство или робот двигались по прямой линии или поворачивали с заданной скоростью.

РЕШЕНИЕ

Эта задача решается с помощью *гироскопа*. Гироскопы выдают сигнал, содержащий информацию о скорости вращения (в отличие от акселерометров, выходной сигнал которых содержит информацию об ускорении объекта). На ранних этапах

[6] EEPROM — электрически стираемое программируемое постоянное запоминающее устройство (ЭСППЗУ).

платформы Arduino большинство недорогих гироскопов выдавали аналоговый сигнал, напряжение которого было пропорционально скорости вращения. Но в настоящее время, когда гироскопы и акселерометры повсеместно используются в смартфонах, легче и дешевле найти модули, совмещающие гироскоп и акселерометр с выходным сигналом, основанным на протоколе I^2C (дополнительная информация по работе с протоколом I^2C приводится в *главе 13*).

Плата Arduino Nano 33 BLE Sense содержит встроенный гироскоп и акселерометр. Более подробная информация по этой плате и датчикам приведена в *разд. 6.1*.

Сравнительно недорогой блок инерциальных измерений MPU-9250 поддерживает девять степеней свободы[7] (9DOF[8]) и хорошо работает с платформой Arduino. Этот модуль предлагается на адаптерной плате многими поставщиками, включая компанию SparkFun (артикул SEN-12762). Для работы с модулем MPU-9250 доступны несколько библиотек Arduino. Подключите модуль MPU-9250 к плате Arduino, как показано на рис. 6.18, и загрузите в плату скетч из листинга 6.28. В скетче используется библиотека Bolder Flight Systems MPU9250, которую нужно установить, используя Менеджер библиотек среды Arduino IDE (подробные инструкции по установке библиотек сторонних разработчиков приводятся в *разд. 16.2*).

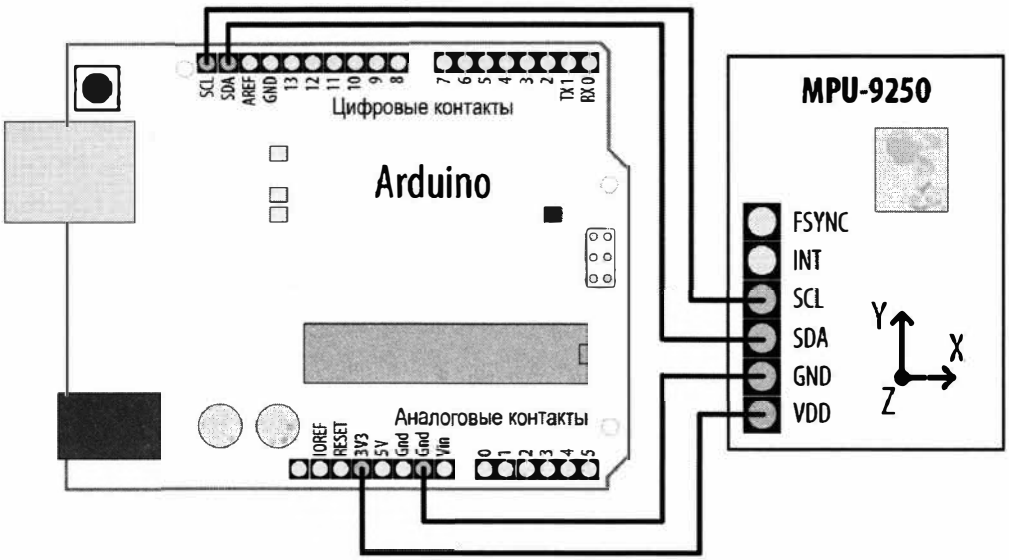


Рис. 6.18. Подключение модуля инерциальных измерений MPU-9250 к плате Arduino

[7] Каждый из трех датчиков модуля (гироскоп, акселерометр и магнитометр) делает измерения в трех пространственных осях (X, Y, Z), то есть имеет три степени свободы, что дает девять степеней свободы для всего модуля в целом.

[8] DOF — от *англ.* Degrees Of Freedom.

Листинг 6.28. Скетч для работы с модулем инерциальных измерений MPU-9250

```
/* Скетч Gyro
* Считывает показания гироскопа и отображает скорость вращения в градусах в секунду
*/

#include "MPU9250.h"

// 0x68 - адрес I2C для модуля инерциальных измерений. В случае
// возникновения проблем с этим адресом, попробуйте адрес 0x69
#define IMU_ADDRESS 0x68

MPU9250 IMU(Wire, IMU_ADDRESS); // Создаем экземпляр объекта IMU

void setup()
{
   Serial.begin(9600);
   while(!Serial);

   // Инициализируем блок инерциальных измерений
   int status = IMU.begin();
   if (status < 0)
   {
      Serial.println("Could not initialize the IMU.");
            // Не удалось инициализировать блок инерциальных измерений
      Serial.print("Error value: "); Serial.println(status);
      while(1); // Останавливаем исполнение программы
   }

   // Задаем полный диапазон измерений гироскопа
   // +/- 500 градусов/секунду
   status = IMU.setGyroRange(MPU9250::GYRO_RANGE_500DPS);
   if (status < 0)
   {
      Serial.println("Could not change gyro range.");
             // Не удалось изменить диапазон измерений гироскопа
      Serial.print("Error value: "); Serial.println(status);
   }
}

void loop()
{
   IMU.readSensor();

   // Получаем скорость вращения в радианах в секунду
   float gx = IMU.getGyroX_rads();
   float gy = IMU.getGyroY_rads();
   float gz = IMU.getGyroZ_rads();
```

```
  // Отображаем скорость вращения в градусах в секунду
  Serial.print("gx:");
  Serial.print(gx * RAD_TO_DEG, 4);
  Serial.print(",gy:");
  Serial.print(gy * RAD_TO_DEG, 4);
  Serial.print(",gz:");
  Serial.print(gz * RAD_TO_DEG, 4);
  Serial.println();
  delay(100);
}
```

Модуль MPU-9250 использует логические уровни I^2C с напряжением 3,3 В, поэтому при использовании его с платой Arduino с питанием 5 В сигнальные линии необходимо подключать через преобразователь логических уровней, чтобы не допустить повреждения контактов SCL и SDA модуля. Интерфейс I^2C и преобразователи логических уровней рассматриваются более подробно в *разд. 13.0*.

Обсуждение работы решения и возможных проблем

В начале скетча подключается библиотека MPU9250 и объявляется экземпляр `IMU` класса этой библиотеки. В функции `setup()` предпринимается попытка инициализировать экземпляр `IMU`. В случае проблем с инициализацией может потребоваться изменить определение адреса `IMU_ADDRESS` на значения `0x69` и/или проверить правильность подключения модуля. После успешной инициализации объекта `IMU` задается новый диапазон измерений гироскопа, равный +/– 500 градусов в секунду.

В функции `loop()` скетч считывает входной сигнал датчика и получает скорость вращения в радианах в секунду. Затем с помощью постоянной Arduino `RAD_TO_DEG` это значение преобразовывается в градусы в секунду. Выходные данные скетча можно наблюдать в мониторе порта или в плоттере по последовательному соединению.

Дополнительная информация

Дополнительная информация по работе с протоколом I^2C приводится в *главе 13*.

Дополнительная информация по подключению устройств с напряжением питания 3,3 В к устройствам с напряжением питания 5 В приводится в *разд. «Использование устройств с напряжением питания 3,3 В с устройствами с напряжением питания 5 В» главы 13*.

Дополнительная информация по модулю инерциальных измерений MPU-9250 приведена в учебном пособии компании SparkFun (**https://oreil.ly/JSXJb**). В этом учебном пособии используется другая библиотека, но принципы работы те же самые.

6.16. Определение ориентации

ЗАДАЧА

Требуется определить ориентацию по сторонам света с помощью *электронного компаса*.

РЕШЕНИЕ

Для решения этой задачи используется компонент *магнитометр* модуля инерциальных измерений MPU-9250, с которым мы познакомились в *разд. 6.15*. Модуль MPU-9250 подключается к плате Arduino так же, как и для решения в *разд. 6.15* (см. рис. 6.18).

Скетч для работы с магнитометром приводится в листинге 6.29.

Прежде чем приступать к работе с магнитометром, его необходимо откалибровать. Соответствующий калибровочный скетч можно найти на веб-странице GitHub: **https://oreil.ly/l58IP**. Полученные калибровочные значения сохраняются в энергонезависимом ЭСППЗУ (EEPROM) микроконтроллера платы Arduino. Их нужно будет загружать из ЭСППЗУ при каждом использовании магнетометра, как показано в скетче решения (листинг 6.30). При использовании модуля IMU с другой микроконтроллерной платой процесс калибровки нужно выполнить снова. Учтите также, что если кроме калибровочных данных магнитометра в ЭСППЗУ сохраняются какие-либо другие данные, необходимо быть внимательным, чтобы не сохранить их в том же участке памяти, что и калибровочные данные.

Листинг 6.29. Скетч для работы с магнитометром

```
/* Скетч Magnetometer
Считывает и отображает показания магнитометра о силе магнитного поля
*/

#include "MPU9250.h"
#include <math.h>
#include "EEPROM.h"

// 0x68 - адрес I2C для модуля инерциальных измерений.
// В случае возникновения проблем с этим адресом, попробуйте адрес 0x69
#define IMU_ADDRESS 0x68

// Измените значение магнитного склонения на соответствующее
// вашему местонахождению.
// Эти данные можно найти на  веб-сайте
// https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml
#define DECLINATION (-14)

MPU9250 IMU(Wire, IMU_ADDRESS); // Создаем экземпляр объекта IMU

void setup()
{
```

```
    int status;

    Serial.begin(9600);

    while (!Serial);

    // Инициализируем блок инерциальных измерений IMU
    status = IMU.begin();
    if (status < 0)
    {
       Serial.println("Could not initialize the IMU.");
            // Не удалось инициализировать блок инерциальных измерений

       Serial.print("Error value: "); Serial.println(status);
       while (1); // Останавливаем исполнение программы
    }
    load_calibration();
}

void loop()
{

    IMU.readSensor();

    // Получаем показания магнитометра по каждой оси в единицах микротесла
    float mx = IMU.getMagX_uT();
    float my = IMU.getMagY_uT();
    float mz = IMU.getMagZ_uT();

    // Код с веб-сайта https://github.com/bolderflight/MPU9250/issues/33
    // Нормализуем данные магнитометра
    float m = sqrtf(mx * mx + my * my + mz * mz);
    mx /= m;
    my /= m;
    mz /= m;

    // Отображаем полученные показания магнитометра
    Serial.print("mx:");
    Serial.print(mx, 4);
    Serial.print(",my:");
    Serial.print(my, 4);
    Serial.print(",mz:");
    Serial.print(mz, 4);
    Serial.println();

    float constrained =
       constrainAngle360(atan2f(-my, mx) + (DECLINATION * DEG_TO_RAD));
    float calcAngle = constrained * RAD_TO_DEG;
    Serial.print(calcAngle);
```

```
    Serial.println(" degrees");
    delay(100);
}

// Код с веб-сайта https://github.com/bolderflight/MPU9250/issues/33
float constrainAngle360(float dta)
{
    dta = fmod(dta, 2.0 * PI);
    if (dta < 0.0)
    dta += 2.0 * PI;
    return dta;
}

// Загружаем калибровочные данные из ЭСППЗУ
// Код с веб-сайта https://github.com/bolderflight/MPU9250/issues/33
void load_calibration()
{
    float hxb, hxs, hyb, hys, hzb, hzs;

    uint8_t eeprom_buffer[24];
    for (unsigned int i = 0; i < sizeof(eeprom_buffer); i++ )
    {
       eeprom_buffer[i] = EEPROM.read(i);
    }
    memcpy(&hxb, eeprom_buffer, sizeof(hxb));
    memcpy(&hyb, eeprom_buffer + 4, sizeof(hyb));
    memcpy(&hzb, eeprom_buffer + 8, sizeof(hzb));
    memcpy(&hxs, eeprom_buffer + 12, sizeof(hxs));
    memcpy(&hys, eeprom_buffer + 16, sizeof(hys));
    memcpy(&hzs, eeprom_buffer + 20, sizeof(hzs));
    IMU.setMagCalX(hxb, hxs);
    IMU.setMagCalY(hyb, hys);
    IMU.setMagCalZ(hzb, hzs);
}
```

При использовании модуля инерциальных измерений с платами с напряжением питания 5 В линии данных модуля необходимо подключать к плате через преобразователь логических уровней. Подробные инструкции по работе с преобразователями логических уровней приводятся в *разд. «Использование устройств с напряжением питания 3,3 В с устройствами с напряжением питания 5 В» главы 13*.

Обсуждение работы решения и возможных проблем

Компонент магнитометр модуля инерциальных измерений предоставляет данные об интенсивности магнитного поля в трех измерениях (X, Y, Z). Эти значения меняются при изменении ориентации магнитометра относительно магнитного поля Земли.

Подобно скетчу решения из *разд. 6.15*, скетч этого решения конфигурирует и инициализирует модуль инерциальных измерений, но вместо считывания показаний его гироскопа считывает показания магнитометра в микротеслах и преобразовывает их в компасный азимут (этот скетч отличается от предыдущего еще и тем, что загружает калибровочные данные из ЭСППЗУ). Для правильной работы скетча модуль инерционных измерений должен быть расположен параллельно горизонтальной поверхности. Необходимо также задать соответствующее вашему местонахождению магнитное склонение, изменив значение переменной `DECLINATION` в начале скетча (для западного склонения используются отрицательные значения, а для восточного — положительные). Данные по магнитному склонению можно получить на веб-сайте центра данных NGDC (**https://oreil.ly/5oDok**).

Полученные с магнитометра показания нормализуются делением показаний каждой оси на квадратный корень суммы квадратов показаний по всем осям. Угол магнитного севера вычисляется добавлением склонения (в радианах) к формуле

```
радиан = arctan2(-my, mx)
```

а затем ограничивается диапазоном в 360 градусов (`2 * PI радиан`) в функции `constrainAngle360()`. Полученный результат преобразовывается в градусы умножением его на константу `RAD_TO_DEG`. Ноль градусов означает азимут на магнитный север.

Чтобы сервомашинка следовала компасному азимуту в диапазоне первых 180 градусов, следует применить метод, рассматриваемый в *разд. «Сервомашинки» главы 8*, но используя переменную `calcAngle` для управления сервомашинкой следующим образом:

```
angle = constrain(calcAngle, 0, 180);
myservo.write(calcAngle);
```

6.17. Определение ускорения

ЗАДАЧА

Требуется реагировать на ускорение объекта — например, определять, когда объект начинает или перестает двигаться. Бывает также нужно определить ориентацию объекта относительно поверхности Земли (измерить ускорение, вызываемое ее притяжением).

РЕШЕНИЕ

Для решения этой задачи используется компонент *акселерометр* модуля инерциальных измерений MPU-9250, с которым мы познакомились в *разд. 6.15*. Модуль MPU-9250 подключается к плате Arduino так же, как и для решения в *разд. 6.15* (см. рис. 6.18).

При использовании модуля инерциальных измерений с платами с напряжением питания 5 В линии данных модуля необходимо подключать к плате через преобразователь логических уровней. Подробные инструкции по работе с преобразователями логических уровней приводятся *в разд. «Использование устройств с напряжением питания 3,3 В с устройствами с напряжением питания 5 В» главы 13*.

В листинге 6.30 приводится простой скетч, получающий показания акселерометра модуля MPU-9250 по осям X, Y и Z и отображающий их в окне монитора порта.

Листинг 6.30. Скетч для получения показаний акселерометра

```
/* Скетч Accelerometer
* Считывается и отображает показания ускорения в м/с/с
*/

#include "MPU9250.h"

// 0x68 - адрес I2C для модуля инерциальных измерений.
// В случае возникновения проблем с этим адресом попробуйте адрес 0x69
#define IMU_ADDRESS 0x68

MPU9250 IMU(Wire, IMU_ADDRESS); // Создаем экземпляр объекта IMU

void setup()
{
   Serial.begin(9600);
   while(!Serial);
   // Инициализируем блок инерциальных измерений
   int status = IMU.begin();
   if (status < 0)
   {
      Serial.println("Could not initialize the IMU.");
            // Не удалось инициализировать блок инерциальных измерений
      Serial.print("Error value: "); Serial.println(status);
      while(1); // Останавливаем исполнение программы
   }
}

void loop()
{
   IMU.readSensor();

   // Получаем показания ускорения по всем трем осям
   float ax = IMU.getAccelX_mss();
   float ay = IMU.getAccelY_mss();
   float az = IMU.getAccelZ_mss();

   // Отображаем ускорение в м/с/с
   Serial.print("ax:"); Serial.print(ax, 4);
```

```
    Serial.print(",ay:"); Serial.print(ay, 4);
    Serial.print(",az:"); Serial.print(az, 4);
    Serial.println();
    delay(100);
}
```

Обсуждение работы решения и возможных проблем

Этот скетч во многом похож на скетч из решения, приведенного в *разд. 6.15*, только вместо скорости вращения он измеряет и отображает ускорение по каждой оси в метрах в секунду за секунду. Обратите внимание, что, даже находясь в неподвижном состоянии, датчик выдает показание ускорения по оси Z в районе 9,8 м/с/с. По крайнем мере, он должен выдавать такое показание при исполнении скетча на планете Земля, сила притяжения которой составляет 9,8 м/с/с. Показание 0 по оси Z означает, что датчик находится в свободном падении. Сила, создающая ускорение в 9,8 м/с/с по оси Z, является механической силой предмета, удерживающего датчик от падения (ваша рука, стол, пол и т. п.). Хотя с точки зрения наблюдателя объект не выглядит ускоряющимся, в действительности он ускоряется относительно свободного падения, которое бы происходило, если бы между датчиком и центром Земли не было бы никакого препятствия: руки, стола, пола и т. п. (отсутствие препятствия между датчиком и центром Земли было бы несколько необычной и определенно нежелательной конфигурацией массы Земли, по крайней мере с точки зрения форм жизни на Земле).

Для получения полезных данных из показаний акселерометра можно использовать методы двух предыдущих решений. Возможно, вам потребуется выполнять проверку на превышение порогового значения, чтобы обработать движение в том или ином направлении (пример такой проверки можно посмотреть в решении из *разд. 6.7*). Также может оказаться полезным обрабатывать входящие данные, используя формулу текущего среднего.

Показания горизонтальных перемещений акселерометра можно использовать напрямую для определения движения в горизонтальном направлении. Но для показаний вертикальных перемещений необходимо учитывать эффект, оказываемый на них силой земного притяжения. Это нечто похожее на учет смещения постоянной составляющей в решении из *разд. 6.8*, но организация этого учета может быть весьма сложной, поскольку изменение направления движения акселерометра может привести к тому, что влияние земного притяжения не будет постоянным для всех замеров.

Обработка получаемых с акселерометров данных может оказаться трудной задачей, особенно при принятии решений о перемещении во времени — определении жестов, а не только местонахождения. Для обработки данных датчиков в реальном времени и распознавания того, как они соотносятся с примерами данных, полученных ранее, сейчас начинают использоваться методы машинного обучения. В настоящее время для реализации таких подходов требуется использовать компьютеры, и их все еще достаточно трудно настроить должным образом, но результаты всего этого могут получиться очень полезные.

Дополнительная информация

Система предсказания жестов на основе примеров (**https://oreil.ly/67i7U**), созданная Дэвидом Меллисом (David Mellis) на основе инструментария распознавания жестов Gesture Recognition Toolkit, является замечательным примером реализации ее с помощью плат Arduino.

Также стоит обратить внимание на проект Wekinator (**https://oreil.ly/qAxef**).

Компания SparkFun предоставляет библиотеку с расширенными возможностями для модуля инерциальных измерений MPU-9250, включающую функции шагомера, определения жеста касания, ориентации и направления. Для работы с этой библиотекой требуется плата Arduino с микроконтроллером SAMD или совместимая плата (**https://oreil.ly/pJyOI**).

ГЛАВА 7

Управление светодиодными устройствами вывода данных

7.0. Введение

Визуальный вывод данных позволяет предоставлять пользователям информацию, и с этой целью платформа Arduino поддерживает широкий круг светодиодных устройств вывода (платформа Arduino также поддерживает работу с графическими дисплеями, которые рассматриваются в *главе 11*). Прежде чем приступить к рассмотрению решений этой главы, мы обсудим возможности цифрового и аналогового вывода плат Arduino и разберемся в том, как Arduino работает со светодиодами. Представленная здесь информация станет хорошей отправной точкой для тех, кто еще не знаком с работой цифрового и аналогового выводов (реализуемых посредством функций `digitalWrite()` и `analogWrite()`) или с использованием светодиодов в схемах. Решения этой главы охватывают обширный диапазон задач: от простых дисплеев с одним светодиодом до создания иллюзии бегущего огня (*разд. 7.7*) и отображения фигур (*разд. 7.9*).

Цифровой вывод

Все контакты платы Arduino, которые можно использовать для ввода цифровых сигналов, можно использовать также и для вывода цифровых сигналов. В *главе 5* приводится обзор схемы расположения контактов платы Arduino. Если вы подзабыли назначение ее контактов и способы подключения к ним периферийных устройств, вы можете освежить свои знания, обратившись к *разд. 5.0*.

Цифровой вывод создает на контакте высокий (5 вольт или 3,3 вольта, в зависимости от платы) или низкий (0 вольт) логический уровень. Этот вывод реализуется функцией `digitalWrite(outputPin, value)` и может использоваться с тем, чтобы включать или выключать подключенное к контакту устройство. Параметр `outputPin` функции обозначает контакт, которым требуется управлять, а параметр `value` — подаваемое на этот контакт значение: `HIGH` (5 или 3,3 вольта) или `LOW` (0 вольт).

Но чтобы контакт мог реагировать на эту команду, для него нужно задать выходной режим работы посредством команды `pinMode(outputPin, OUTPUT)`. Скетч в *разд. 7.1* (см. листинг 7.1) содержит пример использования цифрового вывода.

Аналоговый вывод

В отличие от цифрового сигнала, уровень которого может принимать только два состояния, уровень аналогового сигнала может принимать бесконечно большое количество значений в рамках определенного их диапазона. В качестве примера аналогового сигнала можно назвать сигнал плавного управления уровнем освещения или уровнем звука. Уровнем выходного аналогового сигнала (и соответственно уровнем, например, яркости светодиода, подключенного к соответствующему контакту) можно управлять посредством функции `analogWrite()` языка Arduino.

В действительности сигнал, создаваемый функцией `analogWrite()`, не является по настоящему аналоговым, хотя, как мы вскоре увидим, он может вести себя подобно аналоговому сигналу. Функция `analogWrite()` эмулирует аналоговый сигнал, используя импульсы цифрового сигнала. Этот метод называется *широтно-импульсной модуляцией* (ШИМ).

Принцип работы широтно-импульсной модуляции заключается в варьировании длительности периода включенного состояния цифрового импульса, как показано на рис. 7.1.

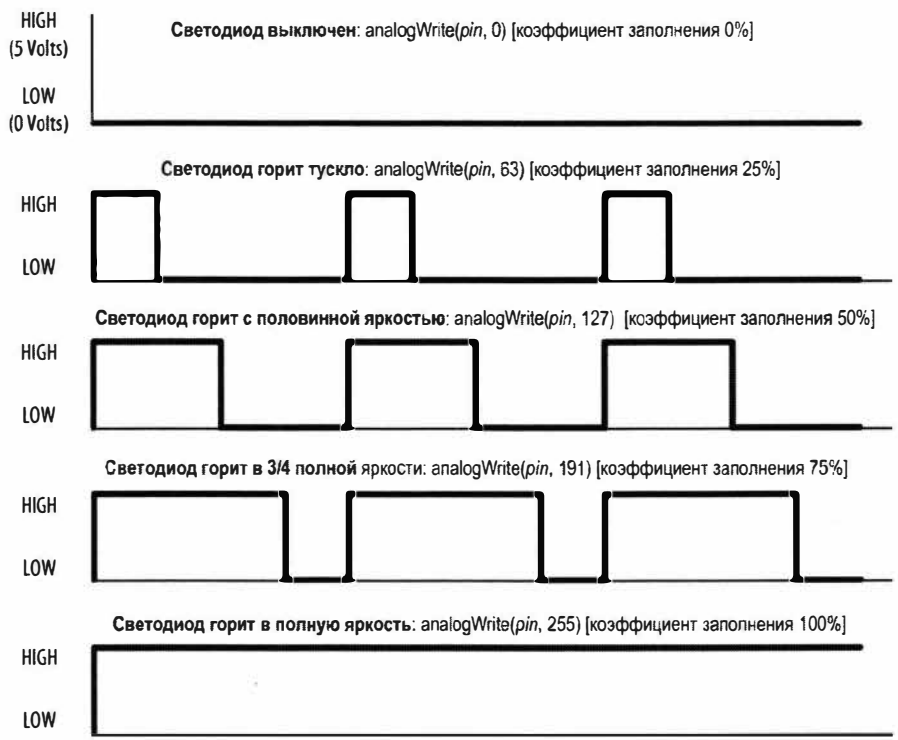


Рис. 7.1. Выходной сигнал ШИМ для разных значений функции `analogWrite()`

Как можно видеть, выходной аналоговый сигнал низкого уровня эмулируется кратковременными цифровыми импульсами. Более высокий уровень эмулируется удерживанием цифрового сигнала во включенном состоянии в течение большего

периода времени, чем в выключенном. При достаточно высокой частоте импульсов (500 импульсов в секунду или более) человеческий глаз не в состоянии различить отдельные события включения и выключения управляемого ими светодиода, а воспринимает всю их последовательность как постоянно включенное состояние светодиода. При этом варьирование коэффициента заполнения цифрового импульса управления светодиодом воспринимается человеческим глазом как изменение яркости свечения светодиода.

Лишь ограниченное количество цифровых контактов платы Arduino могут поддерживать работу с ШИМ-сигналом. На плате Arduino Uno и других платах с микроконтроллером ATmega328 для этой цели можно использовать только контакты: 3, 5, 6, 9, 10 и 11. На платах Arduino Mega работу с ШИМ-сигналом поддерживают контакты со 2-го по 13-й и с 14-го по 46-й. Плата Arduino Nano имеет только пять ШИМ-контактов, а платы Arduino Zero, RedBoard Turbo компании SparkFun и Metro Express M0 компании Adafruit поддерживают ШИМ-сигнал на всех цифровых контактах, за исключением контактов 2 и 7. Во многих последующих решениях задействованы контакты, способные работать как с обычными цифровыми, так и с ШИМ-сигналами, чтобы уменьшить объем изменений, которые нужно внести в схему для использования ее с другими примерами. Если вы решите задействовать другие контакты для ШИМ-сигнала, вместо указанных в примерах, то эти контакты должны поддерживать функцию `analogWrites()`, иначе на них не будет создаваться ШИМ-сигнал. Платы Zero, RedBoard Turbo и Metro Express M0 оснащены цифроаналоговым преобразователем, который создает настоящий аналоговый сигнал, подаваемый на контакт A0 платы. Но этот сигнал предназначен не для управления, например, яркостью светодиода или скоростью электродвигателя (для этих целей лучше подходит ШИМ-сигнал), а для создания звуковых эффектов (см. *разд. 1.8*).

Управление световыми устройствами

Использование выходного цифрового или аналогового сигнала для управления световыми устройствами предоставляет универсальный, эффективный и широко используемый метод обеспечения интерактивного взаимодействия с пользователем. В примерах этой главы всесторонне рассматривается работа с отдельными светодиодами, массивами светодиодов и светодиодными цифровыми дисплеями. Работа с жидкокристаллическими и графическими дисплеями требует иных методов и рассматривается в *главе 11*.

Технические характеристики светодиодов

Светодиод представляет собой особый вид диода — полупроводникового устройства с выводами анода и катода. Когда на аноде светодиода присутствует напряжение большего потенциала, чем на катоде (на величину, называющуюся *прямым напряжением*), светодиод начинает испускать фотоны, т. е. светиться. Вывод анода светодиода обычно имеет большую длину, чем вывод катода, который также часто обозначается плоской гранью на фланце в нижней части линзы светодиода (см. далее

рис. 7.2). Цвет свечения светодиода и точное значение прямого напряжения определяются особенностями конструкции светодиода.

Типичный красный светодиод рассчитан на прямое напряжение величиной приблизительно 1,8 вольта. Если напряжение на аноде этого светодиода не превышает напряжение на катоде на 1,8 вольта, ток через светодиод не будет протекать и светодиод не станет светиться. Когда напряжение на аноде становится на 1,8 вольта более положительным, чем напряжение на катоде, светодиод включается (начинает проводить ток) и, по сути, между его анодом и катодом создается короткое замыкание. Поэтому необходимо ограничивать протекающий через светодиод ток, иначе светодиод раньше или позже перегорит. Как вычислять значение сопротивления токоограничивающих резисторов, показано в *разд. 7.1*.

Чтобы выбрать правильный светодиод для своего приложения, может потребоваться обратиться к справочным листкам (datasheets), чтобы узнать прямое напряжение и максимальный ток рассматриваемых устройств. В табл. 7.1 и 7.2 приведены наиболее важные параметры, на которые следует обращать внимание при изучении справочных листков светодиодов.

Таблица 7.1. *Основные характеристики светодиодов: абсолютные максимальные значения*

Характеристика	Обозначение	Значение	Единицы	Описание
Прямой ток	$I_F(I_{Пр})$	25	мА	Максимальный непрерывный ток светодиода
Пиковый прямой ток (с коэффициентом заполнения 1/10 на частоте 1 КГц)	I_{FP} ($I_{Пр\text{-}Пик}$)	160	мА	Максимальный ток импульсного сигнала (здесь для сигнала с коэффициентом заполнения 10%)

Таблица 7.2. *Основные характеристики светодиодов: электрооптические характеристики*

Характеристика	Обозначение	Значение	Единицы	Описание
Яркость	I_v	2	мкд	Если I_v = 2 мА — яркость при токе 2 мА
	I_v	40	мкд	Если I_v = 20 мА — яркость при токе 20 мА
Угол видимости		120	градусы	Угол испускаемого луча света
Длина волны		620	нм	Доминантная или пиковая длина волны (цвет)
Прямое напряжение	V_F	1,8	вольты	Напряжение на выводах светодиода во включенном состоянии

Контакты плат Arduino Uno, Leonardo и Mega могут выдавать до 40 мА тока. Этого более чем достаточно для питания светодиода средней яркости, но недостаточно для более ярких светодиодов или для нескольких светодиодов, подключенных

к одному контакту. Способ повышения тока для питания светодиодов рассматривается в *разд. 7.3*.

Платы с напряжением питания 3,3 В имеют меньшую токовую возможность. Обратитесь за этой информацией к справочному листку для своей платы, чтобы не превысить максимально допустимое значение потребления тока.

Многоцветные светодиоды содержат два или более светодиода в одном физическом корпусе. У них обычно имеется больше двух выводов для управления отдельными цветами. Такие светодиоды предлагаются во многих вариантах корпуса, и чтобы определить, как подключать их выводы, обратитесь к справочному листку на конкретный светодиод.

Мультиплексирование

Для управления светодиодами, количество которых превышает количество цифровых контактов, можно использовать метод *мультиплексирования*. Этот метод заключается в последовательном включении групп светодиодов (обычно организованных в матрицу строк и столбцов). В *разд. 7.12* показано, как можно управлять 32 отдельными светодиодами четырех 7-сегментных дисплеев (каждый такой дисплей содержит восемь светодиодов, включая светодиод для десятичной точки), используя только 12 цифровых контактов. При этом восемь контактов управляют сегментом для всех четырех дисплеев, а четыре контакта определяют выбор активного дисплея. Перебирая все дисплеи с достаточно высокой скоростью (по крайней мере 25 раз в секунду), вы создаете впечатление непрерывного свечения светодиодов дисплеев. Этот феномен называется *инерционностью зрительного восприятия*.

Версия мультиплексирования, называющаяся *чарлиплексированием*[1], позволяет управлять большим количеством светодиодов, пользуясь *полярностью* этих устройств (полярность означает, что светодиод включается только в том случае, когда напряжение на аноде больше напряжения на катоде) для переключения между двумя светодиодами, меняя полярность питания на обратную.

Максимальный ток контакта

Светодиоды могут потреблять больше тока, чем может предоставить микроконтроллер платы Arduino. Согласно справочному листку на него абсолютная максимальная токовая нагрузка на каждый отдельный контакт микроконтроллера платы Arduino Uno (ATmega328P) не должна превышать 40 мА. При этом общая нагрузка на микроконтроллер по входному и выходному токам не должна превышать 200 мА. Это, например, пять контактов выходного сигнала высокого уровня (выходной ток) и пять контактов входного сигнала низкого уровня (входной ток), где каждый контакт поставляет 40 мА тока. Хорошая практика разработки приложений заключается в умении рассчитывать потребляемый контактами ток с хорошим запасом в пределах абсолютных максимальных значений, чтобы обеспечить наивысшую надежность. Например, лучше всего удерживать ток потребления для отдель-

[1] Метод назван по имени своего создателя Чарли Аллена (Charlie Allen) из компании Maxim.

ного контакта на уровне 30 мА или ниже, чтобы обеспечить большой запас по току. При разработке любительских приложений, которым требуется большее потребление тока, а пониженная надежность является приемлемой, нагрузка на контакт может быть увеличена вплоть до 40 мА при условии, что общее потребление для микроконтроллера не превысит 200 мА исходящего и 200 мА входящего тока.

В *разд. «Как получить больше 40 мА на микроконтроллере Atmega?»* далее в этой главе рассказано, как повысить токовую нагрузку, не прибегая к использованию внешних транзисторов.

В справочном листке указывается величина 40 мА как максимальное абсолютное значение нагрузки на контакт, и большинство инженеров остерегаются создавать нагрузку на контакт, близкую к этому значению. Однако значение 40 мА уже было пересмотрено компанией Atmel, которая утверждает, что контакты могут выдерживать этот ток без каких бы то ни было проблем. В последующих примерах решений подразумевается максимальная нагрузка величиной 40 мА, но в проектах, для которых надежность является первостепенным фактором, имеет смысл для подстраховки понизить это значение до 30 мА. Но при этом следует иметь в виду, что максимальная нагрузка для плат с питанием 3,3 В и даже для некоторых плат с питанием 5 В должна быть еще ниже. В частности, плата Arduino Uno WiFi Rev2 допускает максимальную нагрузку на контакт величиной 20 мА, а плата Zero — всего лишь 7 мА. Если вы используете какую-либо другую плату, проверьте ее максимальный ток нагрузки по ее справочному листку.

7.1. Подключение и использование светодиодов

ЗАДАЧА

Требуется управлять одним или несколькими светодиодами. При этом, чтобы не повредить светодиоды, их необходимо подключить к плате Arduino через токоограничивающие резисторы.

РЕШЕНИЕ

Включение и выключение светодиода посредством платы Arduino не представляет никаких проблем, и эта возможность уже использовалась в ряде решений в предыдущих главах. Например, решение в *разд. 5.1* управляет встроенным светодиодом, подключенным к контакту 13. Здесь же мы рассмотрим общие правила выбора внешних светодиодов для подключения к плате Arduino и работы с ними. На рис. 7.2 показано подключение к плате Arduino трех светодиодов, но скетч решения (листинг 7.1) можно исполнять, подключив хотя бы один или два из них.

На принципиальных схемах в англоязычной литературе катод (cathode) (отрицательный вывод) обозначается буквой k, а не c. Буквой c обозначаются конденсаторы (capacitor).

Скетч в листинге 7.1 последовательно включает и выключает три светодиода с периодом включенного состояния длительностью в секунду.

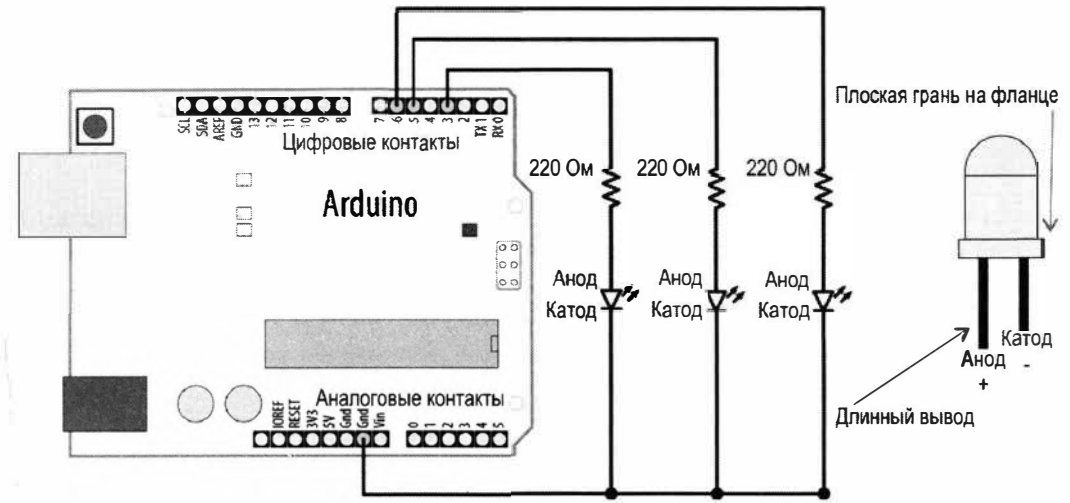


Рис. 7.2. Подключение внешних светодиодов к плате Arduino

Листинг 7.1. Скетч последовательно мигает тремя светодиодами один раз в секунду

```
/*
* Скетч LEDs
* Последовательно включает и выключает три светодиода
*/

const int firstLedPin = 3; // Контакты для подключения светодиодов
const int secondLedPin = 5;
const int thirdLedPin = 6;

void setup()
{
   pinMode(firstLedPin, OUTPUT);  // Задаем выходной режим
                                  // работы для контактов светодиодов
   pinMode(secondLedPin, OUTPUT);
   pinMode(thirdLedPin, OUTPUT);
}

void loop()
{
   // Включаем каждый светодиод на 1000 мс (1 секунду)
   blinkLED(firstLedPin, 1000);
   blinkLED(secondLedPin, 1000);
   blinkLED(thirdLedPin, 1000);
}

// Мигаем светодиодом на заданном контакте в течение duration миллисекунд
void blinkLED(int pin, int duration)
{
```

```
    digitalWrite(pin, HIGH); // Включаем светодиод
    delay(duration);
    digitalWrite(pin, LOW);  // Выключаем светодиод
    delay(duration);
}
```

Контакты для управления светодиодами конфигурируются для работы в режиме вывода в функции `setup()`. В главном цикле `loop()` для каждого из этих контактов вызывается функция `blinkLED()`, которая включает и выключает соответствующий светодиод, устанавливая на контакте высокий уровень в течение одной секунды (1000 мс), а затем снимая его.

Обсуждение работы решения и возможных проблем

Поскольку светодиоды подключены анодами к сигнальным контактам платы Arduino, а катодами к шине «земли», они включаются подачей на соответствующий контакт высокого логического уровня (`HIGH`), а выключаются подачей на него низкого уровня (`LOW`). Логику включения и выключения можно изменить на обратную: включать светодиод низким уровнем (`LOW`), а выключать — высоким (`HIGH`). Для этого катод светодиода нужно подключить к контакту управления, а анод — к шине питания 5 В. Токоограничивающие резисторы можно подключить с любой стороны светодиода: как к катоду, так и к аноду.

Такое подключение светодиодов показано на рис. 7.3. Исполнение скетча решения в этом случае будет иметь обратный эффект: один из светодиодов выключается на одну секунду, а остальные два в это время находятся во включенном состоянии.

Внешние светодиоды необходимо подключать к плате Arduino последовательно с токоограничивающим резистором, иначе они быстро сгорят. Резисторы можно подключать как к аноду, так и к катоду светодиода.

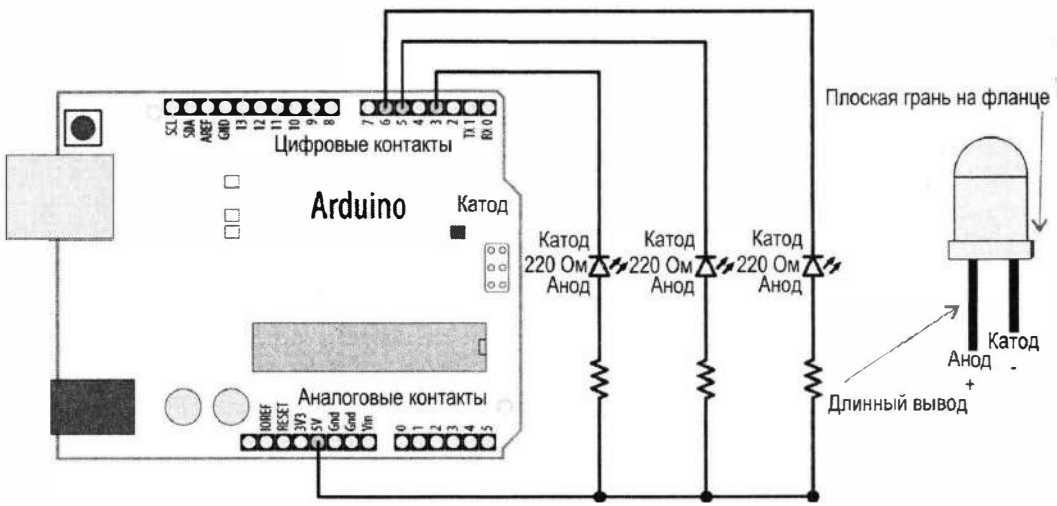


Рис. 7.3. Подключение светодиодов катодами к контактам платы Arduino

Для ограничения протекающего через светодиод тока светодиод подключается к плате Arduino через токоограничивающий резистор. Чтобы рассчитать сопротивление этого резистора, нужно знать напряжение источника питания ($U_П$, обычно 5 В), прямое напряжение ($U_{ПР}$) светодиода и силу тока (I), который нужно пропускать через светодиод.

Для вычисления сопротивления используется формула *закона Ома*:

$$R = (U_П - U_{ПР}) / I. \quad (6.1)$$

Рассмотрим, например, подключение светодиода с прямым напряжением 1,8 В и рабочим током 15 мА к источнику питания напряжением 5 В. Значения для формулы (6.1) будут следующие:

- $U_П$ = 5 В — для платы Arduino с питанием 5 В;
- $V_{ПР}$ = 1,8 В — прямое напряжение светодиода;
- I = 0,015 А (1 миллиампер [мА] равен одной тысячной ампера, следовательно, 15 мА — это 0,015 ампер);

Напряжение на включенном светодиоде:

$$(U_П - V_{ПР}) = 5 \text{ В} - 1{,}8 \text{ В} = 3{,}2 \text{ В}.$$

Отсюда сопротивление последовательного токоограничивающего резистора:

$$3{,}3 \text{ В} / 0{,}015 \text{ А} = 213 \text{ Ом}.$$

Поскольку значение 213 Ом не является стандартным, округляем его до ближайшего стандартного значения: 220 Ом.

На рис. 7.2 токоограничивающий резистор подключен между катодом светодиода и «землей», но его также можно подключить и на другой стороне светодиода — между шиной питания и анодом.

Для контактов плат Arduino Uno и Mega указана максимальная токовая нагрузка величиной 40 мА. Если ваш светодиод потребляет больший ток, чем контакт может предоставить, решение этой задачи вы найдете в *разд. 7.3*.

Дополнительная информация

В *разд. 7.3* показано, как подключать светодиоды с бóльшим потреблением тока, чем может предоставить контакт платы Arduino.

7.2. Управление яркостью светодиода

ЗАДАЧА

Требуется управлять яркостью одного или нескольких светодиодов.

РЕШЕНИЕ

Для решения этой задачи светодиоды нужно подключить к контактам, поддерживающим ШИМ-сигнал. Такое подключение показано на рис. 7.2. Скетч для управ-

ления светодиодами приводится в листинге 7.2. Скетч включает светодиоды на полную яркость, а затем постепенно понижает яркость вплоть до минимальной, после чего процесс, который занимает около пяти секунд, повторяется.

Листинг 7.2. Управления яркостью светодиодов

```
/*
* Скетч LedBrightness
* Управляет яркостью светодиодов, подключенных к ШИМ-контактам
*/

const int firstLed = 3; // Контакты для подключения светодиодов
const int secondLed = 5;
const int thirdLed = 6;

int brightness = 0;
int increment = 1;

void setup()
{
   // Для контактов, на которые выходной сигнал подается
   // функцией analogWrite(), не требуется явно задавать
   // выходной режим работы
}

void loop()
{
   if(brightness > 255)
   {
      increment = -1; // После достижения значения 255 начинаем
                      // обратный счет
   }
   else if(brightness < 1)
   {
      increment = 1; // После достижения значения 0 начинаем прямой счет
   }

   brightness = brightness + increment; // Увеличиваем яркость
               // (или уменьшаем, если значение increment = -1
   // Записываем значение яркости на контакты управления светодиодами
   analogWrite(firstLed, brightness);
   analogWrite(secondLed, brightness);
   analogWrite(thirdLed, brightness );

   delay(10); // 10 мс задержка в каждой итерации цикла означает
              // длительность в 2,55 секунды периода уменьшения
              // или увеличения яркости светодиодов
}
```

Обсуждение работы решения и возможных проблем

Для этого решения используется та же схема подключения светодиодов, что и для предыдущего решения, но здесь выходной сигнал на контакты управления светодиодами подается с помощью функции `analogWrite()`, а не `digitalWrite()`. Функция `analogWrite()` управляет подаваемой на контакт мощностью, используя ШИМ-сигнал, который подробно рассматривается во *введении* в эту главу (*разд. 7.0*).

Скетч повышает и понижает яркость светодиодов, увеличивая и уменьшая значение переменной `brightness` в каждой итерации главного цикла `loop()`, которое затем передается функцией `analogWrite()` на контакты управления светодиодами. Минимальное значение переменной `brightness` равно нулю и соответствует напряжению выходного сигнала 0 В. А максимальное значение равно 255 и соответствует напряжению 5 В (или 3,3 В для плат с напряжением питания 3,3 В).

Рекомендуется ограничить диапазон значений переменной `brightness` диапазоном 0–255, поскольку значения вне этого диапазона могут вызвать неожиданные результаты. Более подробно тема ограничения диапазона значений рассматривается в *разд. 3.5*.

После достижения максимального значения переменной `brightness` оно начинается уменьшаться, т. к знак переменной `increment` меняется с +1 на –1 (добавление –1 к значению эквивалентно вычитанию из него 1).

Дополнительная информация

Во *введении* в эту главу (см. *разд. 7.0*) рассматривается работа аналогового вывода платы Arduino.

Такие платы Arduino, как Due, Zero и MKR1000, могут иметь диапазон значений ШИМ от 0 до 4095, но по умолчанию используют стандартный диапазон от 0 до 255. Переключиться на диапазон более высокого разрешения можно с помощью функции `analogWriteResolution()` (**https://oreil.ly/7_kOi**).

7.3. Работа с мощными светодиодами

ЗАДАЧА

Отдельные контакты плат Arduino Uno и Mega могут выдерживать токовую нагрузку, не превышающую 40 мА. Однако нам требуется управлять светодиодами, потребляющими более мощные токи, чем могут предоставить контакты платы Arduino.

РЕШЕНИЕ

Эта задача решается привлечением для управления сигналом с контакта платы Arduino транзистора, который, в свою очередь, управляет светодиодом, включая и выключая протекающий через него ток. Схема подключения транзистора и свето-

диода к плате Arduino показана на *рис.* 7.4. Для работы с этой схемой можно использовать скетчи из *разд. 7.1* или *7.2* (нужно только обеспечить, чтобы контакт базы транзистора был подключен к тому же контакту, что и светодиод в этих скетчах).

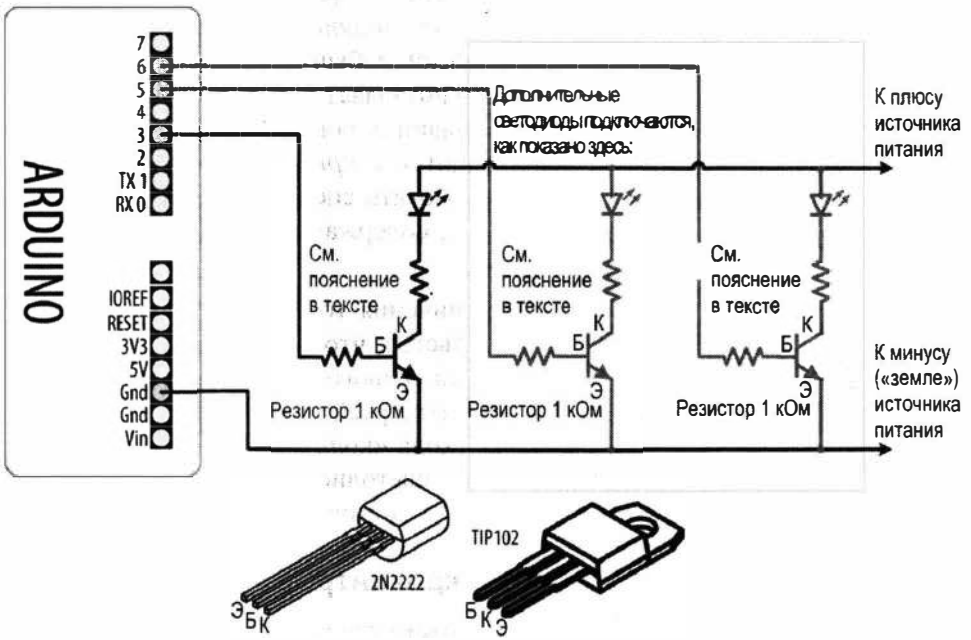


Рис. 7.4. Управление мощными светодиодами с помощью транзисторов

Обсуждение работы решения и возможных проблем

На рис. 7.4 стрелка указывает на подключение к плюсу источника питания. Таким источником питания может быть контакт 5V самой платы Arduino, способный предоставить ток порядка 400 мА при питании платы от компьютера через USB-разъем. При питании платы от внешнего источника питания сила тока, доступная на этом контакте, будет зависеть от силы тока и напряжения используемого источника питания. Избыточное напряжение рассеивается в виде тепла стабилизатором платы Arduino — микросхемой с тремя выводами, которая обычно находится возле разъема для подключения внешнего питания. Прикоснитесь пальцем к этой микросхеме, чтобы проверить, что она не перегревается. Если требуется более сильный ток, чем может предоставить контакт 5V платы Arduino, для питания светодиодов необходимо использовать источник питания, отдельный от источника питания платы. Информация по использованию внешних источников питания приводится в *приложении 3*.

При использовании отдельного источника питания необходимо соединить его шину «земли» с шиной «земли» источника питания платы Arduino.

Подача сигнала высокого уровня на базу транзистора включает транзистор, позволяя току протекать с коллектора на эмиттер. При выключенном транзисторе между этими выводами протекает ничтожный ток. Для включения транзистора используется функция `digitalWrite()`, которая подает сигнал высокого уровня (`HIGH`) на контакт, к которому подключен вывод базы транзистора. Но, подобно подключению светодиода, транзистор к контакту требуется подключать через резистор, чтобы ограничить протекающий через транзистор ток. Обычно используется токоограничивающий резистор номиналом 1 кОм, что позволяет подавать на базу транзистора ток силой 5 мА. Информация по использованию данных справочных листков для выбора необходимого транзистора приводится в *приложении 2*. Для управления мощными нагрузками также можно использовать специальные микросхемы драйверов — например, микросхему ULN2003A, содержащую семь драйверов, способных предоставить ток силой 0,5 А каждый.

При расчете значения резистора для ограничения тока светодиода (см. *разд. 7.1*) следует принять во внимание то обстоятельство, что напряжение источника питания может слегка понизиться вследствие небольшого падения напряжения на транзисторе. Это падение обычно меньше трех четвертей вольта (точное значение можно узнать по напряжению насыщения перехода «коллектор — эмиттер», как показано в *приложении 2*). Для питания мощных светодиодов (1 Ватт и больше) лучше всего использовать стабилизированный по току источник питания.

Как получить больше 40 мА на микроконтроллере Atmega?

Для плат с микроконтроллером ATmega можно соединить параллельно несколько контактов — чтобы повысить максимальный ток выше предела в 40 мА для отдельного контакта (см. *разд. «Максимальный ток контакта»* ранее в этой главе).

На рис. 7.5 показано, как подключить светодиод к двум контактам платы, чтобы обеспечить для него ток 60 мА. Здесь катод светодиода подключается через два токоограничивающих резистора на «землю» через контакты 2 и 7 платы. При подаче на оба эти контакта сигнала низкого уровня (`LOW`) через диод будет протекать ток силой 60 мА. Светодиод подключается к каждому контакту через отдельный рези-

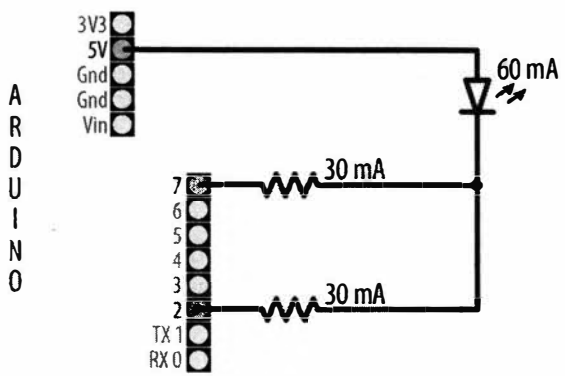


Рис. 7.5. Получение для светодиода тока, большего чем 40 мА

стор. Не пытайтесь использовать один резистор, подключенный одновременно к обоим контактам.

В схеме на рис. 7.5 контакты платы служат источником втекающего тока. Но эту же схему можно использовать с контактами, служащими источником вытекающего тока. Для этого надо поменять местами выводы светодиода: катод подключить к шине «земли» (контакт Gnd), анод — к резисторам, подключенным к контактам платы, а для включения светодиода подавать на контакты сигнал высокого уровня (`HIGH`).

При этом подходе рекомендуется не использовать смежные контакты платы, чтобы свести к минимуму нагрузку на микроконтроллер.

Рассмотренный метод можно использовать только для цифровых сигналов, устанавливаемых посредством функции `digitalWrite()`. Для ШИМ-сигналов, устанавливаемых функцией `analogWrite()`, необходимо использовать подход с применением транзисторов.

Наконец, этот метод не рекомендуется использовать с 32-разрядными платами.

Дополнительная информация

Информация по драйверам со стабилизированным источником тока приводится на веб-странице **https://oreil.ly/XeC63**.

7.4. Управление цветом многоцветного светодиода

ЗАДАЧА

Требуется программно управлять цветом RGB[2]-светодиода.

РЕШЕНИЕ

Многоцветные RGB-светодиоды состоят из трех светодиодов: красного, зеленого и синего цвета, смонтированных в одном корпусе (линзе). Такие RGB-светодиоды могут быть как с общим анодом, когда аноды составляющих светодиодов соединены вместе, так и с общим катодом. Для нашего решения мы воспользуемся RGB-светодиодом с общим анодом, подключенным к плате Arduino, как показано на рис. 7.6: общий вывод анода соединен с контактом 5V платы Arduino, а отдельные выводы катодов подключены к соответствующим контактам платы Arduino. В случае использования RGB-светодиода с общим катодом подключать его следует, как показано на рис. 7.2.

Скетч для работы с этой схемой приводится в листинге 7.3. Скетч непрерывно меняет общий цвет RGB-светодиода, варьируя яркость составляющих его красного, зеленого и синего компонентов.

---

[2] RGB — от *англ.* Red, Green, Blue (красный, зеленый, синий), составляющих цвета светодиода.

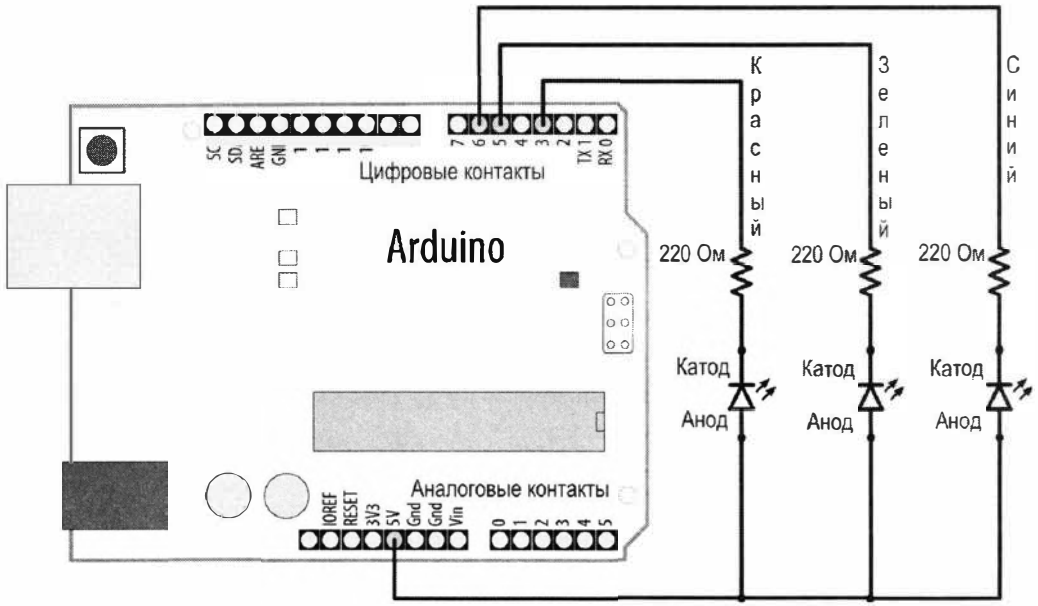


Рис. 7.6. Подключение к плате Arduino RGB-светодиода с общим анодом

Листинг 7.3. Скетч для управления трехцветным RGB-светодиодом

```
/*
* Скетч RGB_LEDs
* Управляет составными цветами RGB-светодиода через ШИМ-контакты
*/

const int redPin = 3;     // Контакты для подключения составных светодиодов
const int greenPin = 5;
const int bluePin = 6;
const bool invert = true; // true для общего анода,
                          // false для общего катода

int color = 0; // Значение от 0 до 255, представляющее цвет в спектре
int R, G, B;   // красный, зеленый и синий компоненты цвета

void setup()
{
   // Для контактов, на которые выходной сигнал подается
   // функцией analogWrite(), не требуется явно задавать
   // выходной режим работы
}

void loop()
{
   int brightness = 255;        // Максимальная яркость равна 255
```

```
    hueToRGB(color, brightness); // Вызов функции для преобразования
                                 // цвета в составляющие RGB-цвета

    // Записываем значения RGB на соответствующие контакты
    analogWrite(redPin, R);
    analogWrite(greenPin, G);
    analogWrite(bluePin, B);

    color++; // Инкрементируем цвет
    if (color > 255)
    color = 0;
    delay(10);
}

// Функция для преобразования цвета в составляющие RGB-цвета
void hueToRGB(int hue, int brightness)
{
    unsigned int scaledHue = (hue * 6);

    // Сегмент от 0 до 5 на цветовом круге
    unsigned int segment = scaledHue / 256;

    // Местонахождение в сегменте
    unsigned int segmentOffset = scaledHue - (segment * 256);

    unsigned int complement = 0;
    unsigned int prev = (brightness * ( 255 - segmentOffset)) / 256;
    unsigned int next = (brightness * segmentOffset) / 256;
    if (invert)
    {
       brightness = 255 - brightness;
       complement = 255;
       prev = 255 - prev;
       next = 255 - next;
    }

    switch (segment)
    {
       case 0: // Красный
          R = brightness;
          G = next;
          B = complement;
          break;

       case 1: // Желтый
          R = prev;
          G = brightness;
          B = complement;
          break;
```

```
      case 2: // Зеленый
         R = complement;
         G = brightness;
         B = next;
         break;

      case 3: // Циан
         R = complement;
         G = prev;
         B = brightness;
         break;

      case 4: // Синий
         R = next;
         G = complement;
         B = brightness;
         break;

      case 5: // Маджента

      default:
         R = brightness;
         G = complement;
         B = prev;
         break;
   }
}
```

Обсуждение работы решения и возможных проблем

Общий цвет RGB-светодиода определяется относительной яркостью его составляющих: красного, зеленого и синего светодиодов. Основная функция скетча `hueToRGB()` преобразовывает значение цвета в диапазоне от 0 до 255 в соответствующий цвет от красного до синего. Спектр видимых цветов часто представляется с помощью цветового круга, содержащего основные и вторичные цвета и их промежуточные оттенки. Сегменты цветового круга, представляющие шесть основных и вторичных цветов, обрабатываются шестью операторами `case`. Код в определенном блоке `case` исполняется в том случае, когда его номер совпадает со значением переменной `segment`, задавая соответствующие значения для красного, зеленого и синего компонентов. Значение 0 переменной `segment` соответствует красному цвету, значение 1 — желтому, значение 2 — зеленому и т. д.

Яркость можно откорректировать, уменьшив значение переменной `brightness`. В следующей строке кода показывается, как установить требуемый уровень яркости, используя переменный резистор или какой-либо другой датчик, подключенный, как показано на рис. 7.14 и 7.18:

```
int brightness = map(analogRead(A0),0,1023,0,255);
```

При варьировании значения аналогового ввода от 0 до 1023 значение переменной `brightness` будет варьироваться в диапазоне от 0 до 255, при этом яркость светодиода станет повышаться при повышении этого значения.

Дополнительная информация

В *разд. 2.16* уже рассматривался пример использования оператора `switch...case`.

7.5. Управление несколькими цветными светодиодами одним контактом платы Arduino

ЗАДАЧА

Требуется управлять цветом нескольких RGB-светодиодов, используя один контакт платы Arduino.

РЕШЕНИЕ

Для решения этой задачи требуется использовать специальный модуль RGB-светодиодов со встроенным микроконтроллером — кольцо NeoPixel. Подключение модуля NeoPixel и потенциометра для управления его цветом к плате Arduino показано на рис. 7.7.

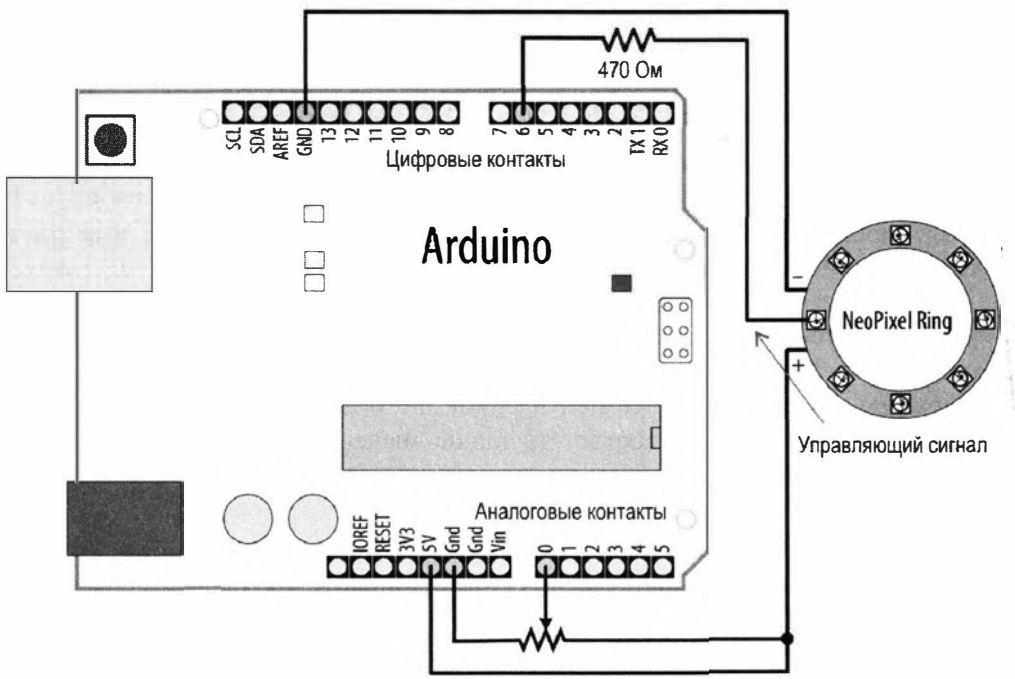


Рис. 7.7. Подключение модуля NeoPixel и потенциометра к плате Arduino

При использовании платы Arduino с питанием 3,3 В как потенциометр, так и положительный вывод модуля NeoPixel следует подключить к контакту 3V3 платы Arduino, а не к контакту 5V.

Скетч для управления этой схемой приводится в листинге 7.4. Для изменения цвета светодиодов модуля в зависимости от положения потенциометра в скетче используется библиотека Adafruit_Neopixels, которую следует установить с помощью Менеджера библиотек среды Arduino IDE.

Листинг 7.4. Работа с модулем RGB-светодиодов NeoPixels

```
/*
* Скетч SimplePixel
* Меняет свет модуля RGB-светодиодов в зависимости от значения
  сигнала датчика
*/

#include <Adafruit_NeoPixel.h>

const int sensorPin = A0; // Номер контакта для подключения датчика
const int ledPin = 6;     // Номер контакта для подключения модуля RGB-светодиодов
const int count = 8;      // Количество светодиодов в модуле

// Объявляем экземпляр объекта модуля RGB-светодиодов
Adafruit_NeoPixel leds = Adafruit_NeoPixel(count, ledPin, NEO_GRB + NEO_KHZ800);

void setup()
{
   leds.begin();    // Инициализируем экземпляр модуля светодиодов
   for (int i = 0; i < count; i++)
   {
      leds.setPixelColor(i, leds.Color(0,0,0)); // Выключаем все светодиоды модуля
   }
   leds.show(); // Обновляем состояние модуля новыми значениями
                // отдельных RGB-светодиодов (все выключены)
}

void loop()
{
   static unsigned int last_reading = -1;

   int reading = analogRead(sensorPin);
   if (reading != last_reading)
   {
      // Если значение входного сигнала изменилось,
      // сопоставляем новое значение диапазону цветов модуля NeoPixel
      unsigned int mappedSensorReading = map(reading, 0, 1023, 0, 65535);
```

```
      // Обновляем состояние отдельных RGB-светодиодов модуля
      // с небольшой задержкой между ними, чтобы создать эффект скольжения
      for (int i = 0; i < count; i++)
      {
         leds.setPixelColor(i, leds.gamma32(
         leds.ColorHSV(mappedSensorReading, 255, 128)));
         leds.show();
         delay(25);
      }
      last_reading = reading;
   }
}
```

Обсуждение работы решения и возможных проблем

Скетч решения управляет модулем NeoPixel компании Adafruit, содержащим восемь RGB-светодиодов, расположенных в виде кольца. Скетч можно использовать с другими версиями модуля NeoPixel, содержащими большее количество RGB-светодиодов (пикселов), присвоив переменной `numOfLeds` соответствующее значение. Но в таком случае следует иметь в виду, что каждый светодиод может потреблять до 60 мА тока (если задать для него белый цвет на полную яркость). Порт USB может запитать до восьми светодиодов, но для большего количества питание на модуль NeoPixel необходимо подавать с отдельного мощного источника питания напряжением 5 В. При этом шину «земли» этого источника питания необходимо подключить к шине «земли» платы Arduino. Кроме того, для плат Arduino с напряжением питания 3,3 В, напряжение питания модуля NeoPixel не должно превышать 3,7 В, поскольку напряжение сигнала данных модуля NeoPixel должно быть примерно равно напряжению питания модуля. При использовании для модуля NeoPixel отдельного источника питания, необходимо установить конденсатор емкостью 1000 мкФ между положительной и отрицательной («землей») шинами источника питания для защиты светодиодов модуля. При этом следует обеспечить правильную полярность подключения этого конденсатора.

В скетче объявляется переменная `leds` объекта модуля NeoPixel с помощью следующей строки кода:

```
Adafruit_NeoPixel leds = Adafruit_NeoPixel(count, ledPin, NEO_GRB + NEO_KHZ800);
```

В результате исполнения этой строки в памяти платы создается структура для хранения цвета каждого светодиода и взаимодействия с модулем. В параметрах объявления указывается количество светодиодов в модуле (`count`), номер контакта платы Arduino (`ledPin`), к которому подключается линия управления модуля, а также тип используемого модуля (`NEO_GRB + NEO_KHZ800`). При использовании другого светодиодного модуля нужно проверить информацию в документации библиотеки и используемого модуля, чтобы узнать, не требуется ли применить иные параметры. Но не будет никакого вреда перепробовать все опции, приведенные в библиотеке, чтобы найти ту, которая будет работать.

Цвет отдельных светодиодов устанавливается с помощью метода (функции) `led.setPixelColor()`. В параметрах функции передается номер требуемого светодиода (нумерация которых начинается со значения 0) и требуемый цвет. Данные на светодиод передаются посредством функции `led.show()`. Прежде чем вызывать функцию `led.show()`, можно изменить значения цвета нескольких светодиодов, которые потом будут применены одновременно с этой функцией. Значения, которые не были изменены, останутся прежними. При создании экземпляра объекта `Adafruit_NeoPixel` все значения цветов инициализируются значением 0.

Библиотека Adafruit NeoPixel содержит собственную функцию `ColorHSV()` для преобразования цвета спектра в значения RGB. Функции передаются три параметра: цвет спектра, насыщенность цвета и яркость, в указанном порядке. Результат, возвращенный функцией `ColorHSV()`, обрабатывается функцией `gamma32()`, которая преобразовывает его, чтобы компенсировать разницу между представлением цвета компьютерами и его восприятием людьми.

Каждый светодиод модуля (так называемый *пиксел*) оснащен контактами для ввода и вывода данных, а также плюса и минуса («земли») питания. Управляющий сигнал с Arduino подается на контакт ввода данных первого светодиода, контакт вывода данных которого подключен к контакту ввода следующего светодиода, и т. д. со всеми светодиодами цепочки. На рынке предлагаются как отдельные пикселы, так и модули из нескольких уже соединенных пикселов, организованных в кольцо или полосу.

Если используемый модуль не поддерживается библиотекой Adafruit

В ранних светодиодных модулях использовалась микросхема WS2811, на смену которой пришли разные другие версии — например, WS2812, WS2812B и APA102. Если ваш светодиодный модуль не поддерживается библиотекой Adafruit, попробуйте использовать библиотеку Fast LED (**http://fastled.io**).

Совместимая с Arduino плата Teensy 3.*x* и более поздние ее версии (**https://www.pjrc.com/teensy**) может управлять восемью модулями и благодаря своему эффективному аппаратному и программному обеспечению способна реализовывать высококачественную анимацию.

Как уже упоминалось ранее, пикселы доступны на рынке как в виде отдельных светодиодов, так и установленные на гибких лентах на разных расстояниях друг от друга (указывается в количестве светодиодов на метр или фут). Компания Adafruit выпускает под торговым именем NeoPixel широкий диапазон светодиодных модулей на печатных платах разных форм-факторов: круги, короткие полоски или панели.

Дополнительная информация

Более подробная информация по модулям NeoPixel приводится в руководстве Uberguide компании Adafruit (**https://oreil.ly/zgAVa**).

Веб-страница библиотеки OctoWS2811 (**https://oreil.ly/yxGBM**) поддерживает работу со светодиодными модулями платы Teensy и содержит удачные схемы под-

ключения источников питания для большого количества светодиодов, а также программу movie2serial на языке Processing, которая извлекает данные из файла видео, указанного в программе, и отображает это видео на панели из светодиодных полос.

7.6. Управление последовательностью светодиодов для создания линейного индикатора

ЗАДАЧА

Требуется создать из нескольких светодиодов линейный индикатор, в котором количество смежных включенных светодиодов соответствует указанному в скетче значению или значению, считанному с какого-либо датчика.

РЕШЕНИЕ

Для решения этой задачи к схеме, приведенной на рис. 7.2, можно добавить дополнительные светодиоды, подключив их к свободным цифровым контактам платы Arduino. Подключение шести светодиодов к последовательным контактам платы показано на рис. 7.8.

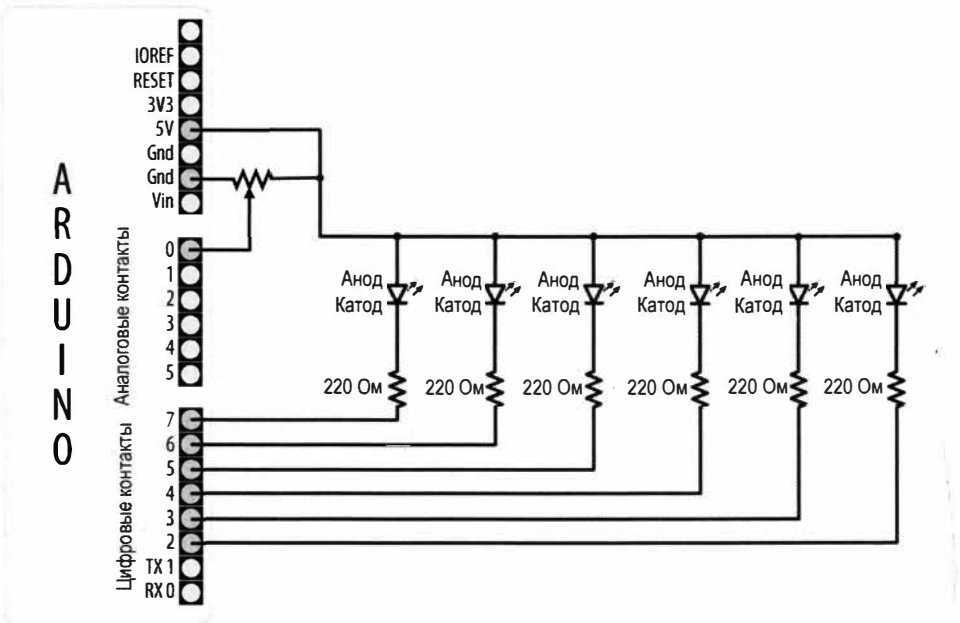


Рис. 7.8. Подключение шести светодиодов к плате Arduino

Для работы с этой схемой используется скетч, приведенный в листинге 7.5. Скетч включает определенное количество смежных светодиодов, точное число которых зависит от значения, считываемого с датчика, подключенного к входному аналоговому контакту (подключение датчика показано на рис. 7.14 и 7.18).

Листинг 7.5. Управление линейным светодиодным индикатором

```
/*
* Скетч Bargraph
*
* Включает последовательность светодиодов, количество которых
пропорционально значению, выдаваемому аналоговым датчиком
* В этом примере используются шесть светодиодов, но их количество можно
изменить, изменив значение переменной NbrLEDs и добавив дополнительные
номера контактов в массив ledPins
*/

const int NbrLEDs = 6;
const int ledPins[] = { 2, 3, 4, 5, 6, 7 };
const int analogInPin = A0; // Входной аналоговый контакт для подключения
                            // потенциометра

// Поменяйте значения следующих двух определений, если катоды
// светодиодов подключены к шине «земли»
#define LED_ON LOW
#define LED_OFF HIGH

int sensorValue = 0; // Считываемое с датчика значение
int ledLevel = 0;    // Значение датчика, преобразованное
                     // в светодиодный "уровень"

void setup()
{
   for (int led = 0; led < NbrLEDs; led++)
   {
      pinMode(ledPins[led], OUTPUT); // Задаем выходной режим работы
                      // для всех контактов управления светодиодами
   }
}

void loop()
{
   sensorValue = analogRead(analogInPin); // Считываем входной аналоговый сигнал
   ledLevel = map(sensorValue, 10, 1023, 0, NbrLEDs); // Сопоставляем
                       // полученное значение количеству светодиодов
   for (int led = 0; led < NbrLEDs; led++)
   {
      if (led < ledLevel)
      {
         digitalWrite(ledPins[led], LED_ON); // Включаем светодиоды ниже уровня
      }
      else
      {
```

```
            digitalWrite(ledPins[led], LED_OFF); // Выключаем светодиоды выше уровня
        }
    }
}
```

Обсуждение работы решения и возможных проблем

Номера контактов, к которым подключаются светодиоды, хранятся в массиве `ledPins`. Чтобы изменить количество управляемых светодиодов, можно добавить или удалить элементы этого массива, но при этом не забудьте обеспечить, чтобы значение переменной `NbrLEDs` было равным количеству элементов массива, которое, в свою очередь, должно быть равным количеству номеров контактов. Вычисление значения переменной `NbrLEDs` можно предоставить компилятору, заменив эту строку кода:

```
const int NbrLEDs = 6;
```

этой:

```
const int NbrLEDs = sizeof(ledPins) / sizeof(ledPins[0]);
```

Функция `sizeof()` возвращает размер (количество байтов) переменной, что в нашем случае является количеством байтов в массиве `ledPins`. Поскольку это массив целых чисел типа `int` (в котором каждый элемент занимает два байта), общее количество байтов массива делится на размер одного элемента (возвращаемый функцией `sizeof(ledPins[0])`), что дает количество элементов массива.

Количество светодиодов, которые нужно включить в ответ на считанное показание датчика, вычисляется функцией `map()`. Код обрабатывает в цикле каждый светодиод, включая его, если значение датчика больше чем номер светодиода. Например, если значение датчика меньше 10, не включается ни один из светодиодов, а если значение датчика равно половине диапазона возможных значений, включается первая половина светодиодов. В идеальном мире минимальное положение потенциометра должно возвращать ноль, но в реальном мире значение при этом положении будет дрейфовать (колебаться в небольших пределах). При максимальном значении сигнала датчика включаются все светодиоды линейного индикатора. Если при максимальном положении потенциометра последний светодиод мигает, уменьшите второй параметр функции `map()` с 1023 до 1000 или около того.

На рис. 7.8 аноды всех светодиодов соединены вместе (такая конфигурация называется *общим анодом*), а катоды подключены к управляющим контактам. Для включения светодиода в такой конфигурации на контакт управления необходимо подать логический сигнал низкого (`LOW`) уровня. Если же светодиоды подключены к управляющим контактам анодами (как показано на рис. 7.2), а вместе соединены их катоды (такая конфигурация называется *общим катодом*), для включения светодиода на контакт управления необходимо подать логический сигнал высокого уровня (`HIGH`). В скетче решения используются константы `LED_ON` и `LED_OFF`, чтобы упростить задачу включения и выключения светодиодов при использовании любой из этих двух конфигураций подключения светодиодов. Чтобы изменить скетч под конфи-

гурацию с общим катодом, нужно просто поменять значения этих констант, как показано в следующем фрагменте кода:

```
const bool LED_ON = HIGH; // Высокий уровень HIGH включает светодиоды
                          // в конфигурации с общим катодом
const bool LED_OFF = LOW;
```

Скорость выключения светодиодов можно понизить, чтобы, например, имитировать затухание столбчатого индикатора громкости. В листинге 7.6 приводится версия скетча, реализующая эффект затухания индикатора при понижении уровня входящего сигнала.

Листинг 7.6. Имитация эффекта затухания столбчатого индикатора

```
/*
* Скетч LED bar graph - версия с затуханием
*/

const int ledPins[] = {2, 3, 4, 5, 6, 7};
const int NbrLEDs = sizeof(ledPins) / sizeof(ledPins[0]);
const int analogInPin = A0; // Входной аналоговый контакт для подключения
                            // потенциометра
const int decay = 10;       // Увеличение этого значения уменьшает скорость затухания

// Поменяйте значения следующих двух определений, если катоды
// светодиодов подключены к шине "земли"
#define LED_ON LOW
#define LED_OFF HIGH

// Значение скорости затухания
int storedValue = 0;

void setup()
{
   for (int led = 0; led < NbrLEDs; led++)
   {
      pinMode(ledPins[led], OUTPUT); // Задаем выходной режим работы
                     // для всех контактов управления светодиодами
   }
}

void loop()
{
   int sensorValue = analogRead(analogInPin);   // Считываем входной аналоговый сигнал
   storedValue = max(sensorValue, storedValue); // Если значение
     // с датчика более высокое, чем сохраненное, используем его
   int ledLevel = map(storedValue, 10, 1023, 0, NbrLEDs); // Сопоставляем
                            // полученное значение количеству светодиодов
```

```
   for (int led = 0; led < NbrLEDs; led++)
   {
      if (led < ledLevel )
      {
         digitalWrite(ledPins[led], LED_ON); // Включаем светодиоды
                                             // ниже полученного уровня
      }
      else
      {
         digitalWrite(ledPins[led], LED_OFF); // Выключаем светодиоды выше уровня
      }
   }
   storedValue = storedValue - decay; // Значение затухания
   delay(10);          // Пауза в 10 мс перед следующей итерацией цикла
}
```

Затухание реализуется строкой кода, которая содержит функцию `max()`. Эта функция возвращает большее из двух значений: значение `sensorValue`, полученное с датчика, или сохраненное значение датчика `storedValue`, к которому было применено затухание. Если значение датчика `sensorValue` более высокое, чем значение с наложенным затуханием `storedValue`, оно сохраняется как `storedValue`. В противном случае значение `storedValue` уменьшается на постоянную затухания `decay` при каждой итерации цикла (выполняющейся с периодом в 10 мс, заданным функцией `delay()`). Увеличение значения постоянной `decay` уменьшает время затухания светодиодов до полного их выключения.

Светодиодный индикатор можно также реализовать, используя светодиодный модуль NeoPixel (листинг 7.7).

Листинг 7.7. Скетч для реализации светодиодного индикатора на модуле NeoPixel

```
/*
* Скетч PixelBarGraph.ino
* Считанное с датчика значение определяет количество включенных светодиодов
*/

#include <Adafruit_NeoPixel.h>

const int sensorPin = A0; // Номер аналогового контакта для подключения датчика
const int ledsPin = 2;    // Номер контакта для подключения
                          // модуля RGB-светодиодов

const int numOfLeds = 16; // Количество светодиодов в модуле

// Константы для автоматического сопоставления значений датчика
const int minReading = 0;
const int maxReading = 1023;
```

```
// Объявляем экземпляр объекта модуля RGB-светодиодов
Adafruit_NeoPixel leds = Adafruit_NeoPixel(numOfLeds, ledsPin, NEO_GRB + NEO_KHZ800);

void setup()
{
   leds.begin(); // Инициализируем экземпляр модуля светодиодов
   leds.setBrightness(25);
}

void loop()
{
   int sensorReading = analogRead(A0);
   int nbrLedsToLight = map(sensorReading, minReading, maxReading, 0, numOfLeds);

   for (int i = 0; i < numOfLeds; i++)
   {
      if ( i < nbrLedsToLight)
      leds.setPixelColor(i, leds.Color(0, 0, 255)); // Синий
   else
      leds.setPixelColor(i, leds.Color(0, 255, 0)); // Зеленый
   }
   leds.show();
}
```

Дополнительная информация

Функция `max()` подробно рассматривается в *разд. 3.6*.

В *разд. 5.6* приводится дополнительная информация по считыванию выходного сигнала датчика с помощью функции `analogRead()`.

В *разд. 5.7* подробно рассматривается функция `map()`.

В *разд. 12.1* и *12.2* показано, как реализовать более точное время затухания. Общее время исполнения одной итерации главного цикла `loop()` в действительности занимает больше чем 10 мс, задаваемых функцией `delay()`, потому что для исполнения кода цикла требуется дополнительная миллисекунда или около того.

7.7. Управление последовательностью светодиодов для создания эффекта бегущего огня

ЗАДАЧА

Требуется включать и выключать последовательность светодиодов, чтобы создать эффект бегущего огня. Этот эффект использовался в телешоу «Knight Rider» («Рыцарь дорог») и «Battlestar Gallactica» («Звездный крейсер “Галактика”»). Создателем обоих этих телешоу был Глен А. Ларсон (Glen A. Larson), поэтому этот эффект еще называется *сканером Ларсона* (Larson scanner).

РЕШЕНИЕ

Для решения этой задачи используется схема, показанная ранее на рис. 7.8, а для управления ею — скетч из листинга 7.8.

Листинг 7.8. Скетч для реализации эффекта бегущего огня

```
/* Скетч Chaser */

const int NbrLEDs = 6;
const int ledPins[] = {2, 3, 4, 5, 6, 7};
const int wait_time = 30;

// Поменяйте значения следующих двух определений, если катоды
// светодиодов подключены к шине "земли"
#define LED_ON LOW
#define LED_OFF HIGH

void setup()
{
   for (int led = 0; led < NbrLEDs; led++)
   {
      pinMode(ledPins[led], OUTPUT);
   }
}

void loop()
{
   for (int led = 0; led < NbrLEDs - 1; led++)
   {
      digitalWrite(ledPins[led], LED_ON);
      delay(wait_time);
      digitalWrite(ledPins[led + 1], LED_ON);
      delay(wait_time);
      digitalWrite(ledPins[led], LED_OFF);
      delay(wait_time * 2);
   }
   for (int led = NbrLEDs - 1; led > 0; led--)
   {
      digitalWrite(ledPins[led], LED_ON);
      delay(wait_time);
      digitalWrite(ledPins[led - 1], LED_ON);
      delay(wait_time);
      digitalWrite(ledPins[led], LED_OFF);
      delay(wait_time * 2);
   }
}
```

Обсуждение работы решения и возможных проблем

Код этого решения похож на код решения из *разд. 7.6*, с той лишь разницей, что контакты управления светодиодами включаются и выключаются не в зависимости от значения считанного с датчика сигнала, а в установленной последовательности. Код содержит два цикла `for`. В первом цикле эффект перемещения огня слева направо реализуется последовательным включением и выключением светодиодов слева направо. Цикл начинается с первого (самого левого) светодиода и по одному включает и выключает следующие светодиоды, пока не будет достигнут и включен самый правый светодиод. Затем начинает исполняться второй цикл, который включает и выключает светодиоды справа налево, начиная с самого правого и перемещаясь по одному светодиоду влево, пока не будет достигнут первый (самый левый) светодиод. Длительность задержки задается переменной `wait_time`, которую, возможно, придется подобрать, чтобы достичь наиболее удовлетворительного эффекта.

7.8. Управления светодиодной матрицей с использованием мультиплексирования

ЗАДАЧА

Требуется управлять матрицей светодиодов, используя для этого как можно меньшее количество контактов платы Arduino.

РЕШЕНИЕ

В решении используется квадратная светодиодная матрица из 64 светодиодов, аноды которых соединены в строки, а катоды — в столбцы (как в матрице компании Jameco, артикул 2132349). Подключение светодиодной матрицы к плате Arduino показано на рис. 7.9, а в листинге 7.9 приводится скетч для управления ею (двухцветные светодиодные матрицы могут быть более доступны, и в них можно использовать только один цвет, если это все, что требуется).

Это решение потребляет сравнительно большой ток и подходит только для реализации на плате Arduino Uno и других платах с микроконтроллером ATmega 328. Платы Arduino Uno WiFi Rev2 и Nano Every, а также большинство (если не все) 32-разрядных плат не могут безопасно предоставить достаточный ток для питания всех этих светодиодов. Более подходящее решение для таких плат рассматривается в разд. *7.10* и *7.14*.

Листинг 7.9. Управление светодиодной матрицей

```
/*
* Скетч matrixMpx
*
* Последовательно включает все светодиоды матрицы 8×8, начиная
  с первого столбца и первой строки
```

```
* Для управления 64 светодиодами матрицы, задействовав всего 16 контактов,
  применяется мультиплексирование
*/

const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[] = {10,11,12,A1,A2,A3,A4,A5};

int pixel = 0;       // Номера светодиода в матрице
int columnLevel = 0; // Столбец матрицы, соответствующий номеру светодиода
int rowLevel = 0;    // Строка матрицы, соответствующая номеру светодиода

void setup()
{
   for (int i = 0; i < 8; i++)
   {
      pinMode(columnPins[i], OUTPUT); // Задаем выходной режим работы
                        // для всех контактов управления светодиодами
      pinMode(rowPins[i], OUTPUT);
   }
}

void loop()
{
   pixel = pixel + 1;
   if(pixel > 63)
   pixel = 0;

   columnLevel = pixel / 8; // Сопоставляем количеству столбцов
   rowLevel = pixel % 8;    // Берем дробную часть
   for (int column = 0; column < 8; column++)
   {
      digitalWrite(columnPins[column], LOW); // Подключаем этот столбец на "землю"
      for(int row = 0; row < 8; row++)
      {
         if (columnLevel > column)
         {
            digitalWrite(rowPins[row], HIGH); // Подключаем все
                                              // светодиоды в строке к +V
         }
         else if (columnLevel == column && rowLevel >= row)
         {
            digitalWrite(rowPins[row], HIGH);
         }
         else
         {
            digitalWrite(columnPins[column], LOW); // Выключаем все
                                                   // светодиоды в этой строке
         }
```

```
        delayMicroseconds(300); // Задержка дает для 64 светодиодов
                                // период кадра длительностью в 20 мс
        digitalWrite(rowPins[row], LOW); // Выключаем светодиод
      }

      // Отключаем этот столбец от шины "земли"
      digitalWrite(columnPins[column], HIGH);
    }
}
```

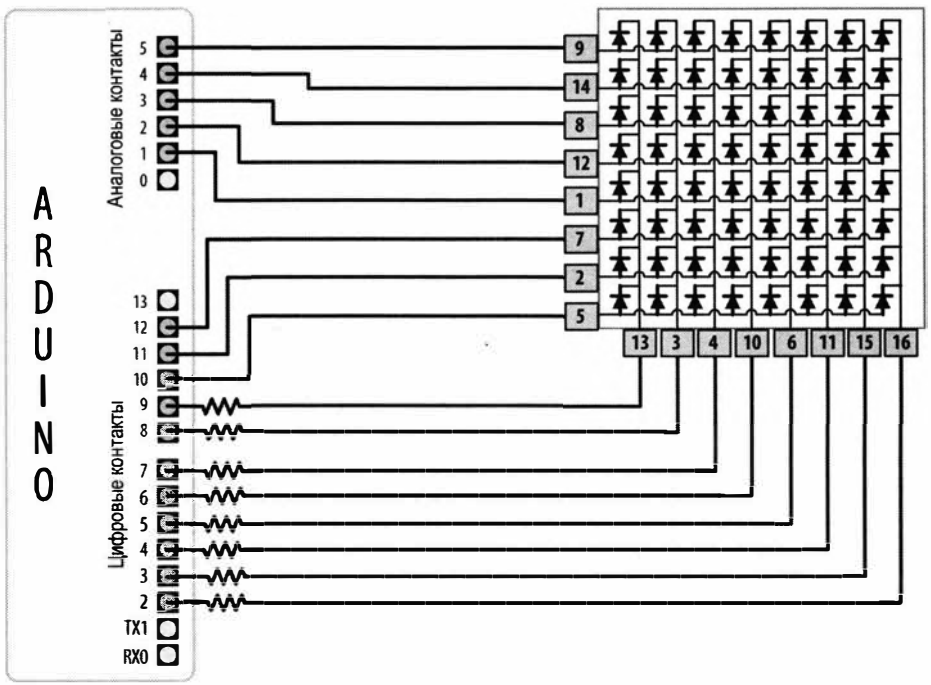


Рис. 7.9. Подключение светодиодной матрицы к плате Arduino

На рис. 7.9 показано подключение светодиодной матрицы 8×8 компании Jameco (актикул 2132349), которая представляет собой распространенный форм-фактор для матриц этого типа. Но для матричных светодиодных дисплеев не существует стандартной схемы контактов, поэтому в случае использования другой светодиодной матрицы информацию по ее схеме контактов ищите в ее справочном листке. Подключите строки анодов и столбцы катодов матрицы к контактам платы Arduino, как показано на рис. 7.9 или 7.16, но используйте номера контактов матрицы с ее справочного листка.

На рис. 7.9 показана логическая организация контактов матрицы, соответствующих столбцам и строкам. Номера контактов на рис. 7.9 также соответствуют их физическому расположению на матрице.

Обычно нумерация контактов следует U-образному шаблону, начинаясь слева вверху (контакты с 1-го по 8-й сверху вниз по левому столбцу контактов и контак-

ты 9–16 снизу вверх по правому столбцу контактов). Сложность состоит в том, чтобы ориентировать устройство таким образом, чтобы контакт 1 находился слева вверху. Определить контакт 1 можно по выемке, обычно в виде небольшой точки, расположенной, скорее всего, на боковой стороне корпуса компонента. Но в случае каких-либо сомнений по этому вопросу, обратитесь за требуемой информацией к справочному листку на компонент.

Обсуждение работы решения и возможных проблем

Значение сопротивлений резисторов необходимо выбрать таким, чтобы ток контакта не превышал 40 мА для платы Arduino Uno (и других плат на микроконтроллере ATmega328). Не применяйте это решение с платами с напряжением питания 3,3 В или с другими платами, контакты которых не могут обеспечить ток величиной 40 мА. Поскольку через один контакт столбца может протекать ток для всех восьми светодиодов, максимальный ток для каждого светодиода должен составлять одну восьмую 40 мА, или 5 мА. Каждый светодиод типичной небольшой матрицы имеет прямое напряжение величиной приблизительно 1,8 вольта. Вычислив сопротивление, дающее ток 5 мА при прямом напряжении 1,8 В, получаем значение 640 Ом, ближайшим стандартным значением к которому будет номинал 680 Ом. При использовании другой светодиодной матрицы посмотрите прямое напряжение ее светодиодов в ее справочном листке. Каждый столбец матрицы подключен через последовательный токоограничивающий резистор к цифровому контакту платы Arduino. Когда на контакт управления столбца матрицы подается сигнал низкого уровня, а на контакт управления строкой сигнал высокого уровня, происходит включение светодиода на пресечении этих столбца и строки. Все светодиоды с высоким уровнем на управляющем контакте столбца или низким уровнем на управляющем контакте строки будут выключены.

Цикл `for` сканирует каждую строку и столбец и последовательно включает светодиоды, пока все они не будут включены. Цикл начинает с первого столбца и первой строки и инкрементирует счетчик строк, пока не будут включены все светодиоды этой строки, после чего переходит к следующему столбцу, и т. д., пока не будут включены все светодиоды матрицы.

Вместо последовательного включения светодиодов матрицы можно включать количество светодиодов в ней, пропорциональное значению сигнала, считанного с датчика, подключенного, как показано на рис. 5.7, внеся следующие изменения в скетч.

Первым делом закомментируйте или удалите следующие три строки кода в цикле `loop()`:

```
pixel = pixel + 1;
if(pixel > 63)
   pixel = 0;
```

Вместо них вставьте следующий код, который считывает значение сигнала датчика, подаваемого на аналоговый контакт A0, и масштабирует его к количеству пикселов (светодиодов) в диапазоне от 0 до 63:

```
int sensorValue = analogRead(0);          // Считываем входной аналоговый сигнал
pixel = map(sensorValue, 0, 1023, 0, 63); // Масштабируем значение
                                  // датчика к количеству пикселов (светодиодов) матрицы
```

В качестве датчика можно использовать простой потенциометр, как показано на рис. 5.7. Количество включенных светодиодов должно быть пропорциональным считанному с датчика значению.

Не обязательно включать одновременно все светодиоды в строке матрицы. Скетч в листинге 7.10 включает светодиоды по одному.

Листинг 7.10. Включение светодиодов матрицы по одному

```
/*
* Скетч matrixMpx, по одному светодиоду за раз
*
* Последовательно включает по одному все светодиоды матрицы 8×8, начиная
с первого столбца и первой строки
* Для управления 64 светодиодами матрицы, задействовав всего
  16 контактов, применяется мультиплексирование
*/

const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[] = {10,11,12,A1,A2,A3,A4,A5};

int pixel = 0;      // Переменная номера светодиода в матрице,
                    // значение от 0 до 63

void setup()
{
   for (int i = 0; i < 8; i++)
   {
      pinMode(columnPins[i], OUTPUT); // Задаем выходной режим работы
                        // для всех контактов управления светодиодами
      pinMode(rowPins[i], OUTPUT);
      digitalWrite(columnPins[i], HIGH);
   }
}

void loop()
{
   pixel = pixel + 1;
   if(pixel > 63)
   pixel = 0;

   int column = pixel / 8; // Сопоставляем количеству столбцов
   int row = pixel % 8;    // Берем дробную часть
```

```
    digitalWrite(columnPins[column], LOW); // Подключаем этот столбец на "землю"
    digitalWrite(rowPins[row], HIGH);      // Подаем на эту строку высокий уровень

    delay(125); // Берем короткую паузу

    digitalWrite(rowPins[row], LOW);        // Подаем на эту строку низкий уровень
    digitalWrite(columnPins[column], HIGH); // Подаем на столбец высокий уровнь

}
```

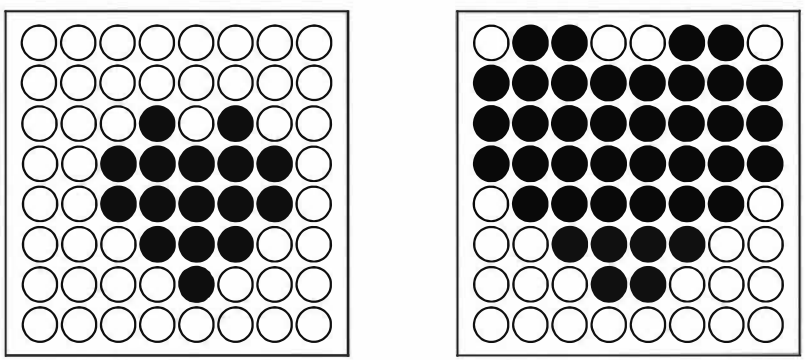
7.9. Вывод изображения на светодиодную матрицу

ЗАДАЧА

Требуется отобразить на светодиодной матрице одно или несколько изображений. Например, создать анимацию, отображая в быстрой последовательности ряд изображений.

РЕШЕНИЕ

Для решения этой задачи используется светодиодная матрица, подключенная по схеме, приведенной на рис. 7.9, и скетч из листинга 7.11. Скетч создает эффект бьющегося сердца, периодически включая светодиоды матрицы, организованные в виде сердца. Для каждого биения сначала включается небольшое изображение сердца, а затем большое (рис. 7.10).



Маленькое изображение сердца | Большое изображение сердца

Рис. 7.10. Для анимации каждого биения отображаются два изображения сердца разного размера

Листинг 7.11. Анимация бьющегося сердца

```
/*
* Скетч matrixMpxAnimation
* Попеременно отображает два изображения сердца разных размеров,
  создавая анимацию бьющегося сердца
*/
```

```
// Изображения сердца сохранены в виде битовых матриц,
   где каждый бит соответствует светодиоду
// 0 обозначает выключенный светодиод, 1 – включенный

byte bigHeart[] =
{
   B01100110,
   B11111111,
   B11111111,
   B11111111,
   B01111110,
   B00111100,
   B00011000,
   B00000000
};

byte smallHeart[] =
{
   B00000000,
   B00000000,
   B00010100,
   B00111110,
   B00111110,
   B00011100,
   B00001000,
   B00000000
};

const int columnPins[] = { 2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[] = {10,11,12,A1,A2,A3,A4,A5};

void setup()
{
   for (int i = 0; i < 8; i++)
   {
      pinMode(rowPins[i], OUTPUT);       // Задаем выходной режим работы
                                 // для всех контактов управления светодиодами
      pinMode(columnPins[i], OUTPUT);
      digitalWrite(columnPins[i], HIGH); // Отключаем все контакты
                                         // столбцов от шины "земли"
   }
}

void loop()
{
   int pulseDelay = 800 ; // Пауза в миллисекундах между биениями
```

```
   show(smallHeart, 80);  // Отображаем малое изображение сердца в течение 80 мс,
   show(bigHeart, 160);   // а за ним большое в течение 160 мс
   delay(pulseDelay);     // Между биениями не отображаем ничего
}

// Отображаем кадр изображения, хранящийся в массиве, на который
// указывает параметр image. Кадр повторяется в течение указанного
// периода duration в миллисекундах.
void show(byte * image, unsigned long duration)
{
   unsigned long start = millis();      // Начинаем отсчет времени для анимации
   while (start + duration > millis()) // Итерируем цикл в течение времени duration
   {
      for(int row = 0; row < 8; row++)
      {
         digitalWrite(rowPins[row], HIGH); // Подаем на контакт строки
                                           // высокий уровень (+5 вольт)
         for(int column = 0; column < 8; column++)
         {
            bool pixel = bitRead(image[row],column);
            if(pixel == 1)
            {
               digitalWrite(columnPins[column], LOW); // Подаем
                           // на контакт столбца низкий уровень
            }
            delayMicroseconds(300);  // Небольшая пауза для каждого светодиода
            digitalWrite(columnPins[column], HIGH); // Подаем на столбец
                                                    // высокий уровень
         }
         digitalWrite(rowPins[row], LOW); // Подаем на строки низкий уровень
      }
   }
}
```

Обсуждение работы решения и возможных проблем

В этом решении столбцы и строки мультиплексируются подобно решению из *разд. 7.8*, но здесь значение сигнала, подаваемого на светодиод, основано на изображениях, сохраненных в массивах `bigHeart` и `smallHeart`. Каждый элемент массива представляет пиксел (светодиод) матрицы, а каждая строка массива представляет строку светодиодной матрицы. Строка массива содержит восемь битов в двоичном формате, обозначенном заглавной буквой `B` (binary, двоичный) в начале каждой строки. Значение бита 1 означает, что соответствующий светодиод должен быть включен, а 0 — выключен. Эффект анимации создается отображением этих двух изображений в быстрой последовательности.

Функция `loop()` выдерживает короткую паузу (800 мс) между биениями, а затем вызывает функцию `show()` сначала с массивом `smallHeart`, а затем с массивом

bigHeart. Функция show() обрабатывает каждый элемент во всех строках и столбцах, включая светодиод, если установлен соответствующий бит. При этом значение каждого бита определяется с помощью функции bitRead() (которая подробно рассматривается в *разд. 2.20*).

Короткая пауза между обработкой пикселов в 300 микросекунд предоставляет достаточно времени, чтобы глаз мог воспринять состояние светодиода. Временны́е параметры выбраны таким образом, чтобы каждое изображение отображалось достаточно часто (50 раз в секунду), — тогда не будет заметно мигание.

В листинге 7.12 приводится фрагмент кода, который изменяет частоту биений сердца в зависимости от значения считанного с датчика сигнала. В качестве датчика можно использовать простой потенциометр, как показано на рис. 5.7. Светодиодная матрица подключается так же, как и для основного решения, и используется тот же скетч, в котором только функция loop() заменена кодом из листинга 7.12.

Листинг 7.12. Код для управления частотой биения сердца на основе сигнала датчика

```
void loop()
{
   int sensorValue = analogRead(A0); // Считываем входной аналоговый сигнал
   int pulseRate = map(sensorValue,0,1023,40,240); // Преобразовываем
                 // полученное значение в количество биений в секунду
   int pulseDelay = (60000 / pulseRate); // Пауза в миллисекундах между биениями

   show(smallHeart, 80); // Отображаем малое изображение сердца в течение 100 мс
   show(bigHeart, 160);  // а за ним большое в течение 200 мс
   delay(pulseDelay);    // Между биениями не отображаем ничего
}
```

Эта версия кода рассчитывает паузу между биениями, используя функцию map() (см. *разд. 5.7*), которая преобразовывает значения датчика в количество биений в минуту. При вычислении не принимается во внимание время для отображения сердца, но точность временны́х расчетов можно повысить, отняв от полученного результата 240 мс (80 мс плюс 160 мс для двух изображений).

Дополнительная информация

В *разд. 7.13* и *7.14* рассматривается использование для управления светодиодами сдвиговых регистров, позволяющих уменьшить количество контактов платы Arduino, необходимых для управления светодиодной матрицей.

В *разд. 12.1* и *12.2* приводится дополнительная информация по управлению временем с помощью функции millis().

7.10. Управление светодиодной матрицей с использованием чарлиплексирования

ЗАДАЧА

Требуется управлять светодиодной матрицей, задействовав для этого как можно меньшее количество контактов платы Arduino.

РЕШЕНИЕ

Одно из возможных решений этой задачи — использование *чарлиплексирования*. Чарлиплексирование — это особый вид мультиплексирования, позволяющий увеличить количество устройств (светодиодов), которыми можно управлять одной группой контактов. В листинге 7.13 приводится скетч для управления посредством чарлиплексирования шестью светодиодами, задействовав всего лишь три контакта платы Arduino. Подключение светодиодов показано на рис. 7.11 (методика вычисления значений токоограничивающих резисторов для светодиодов приводится в *разд. 7.1*).

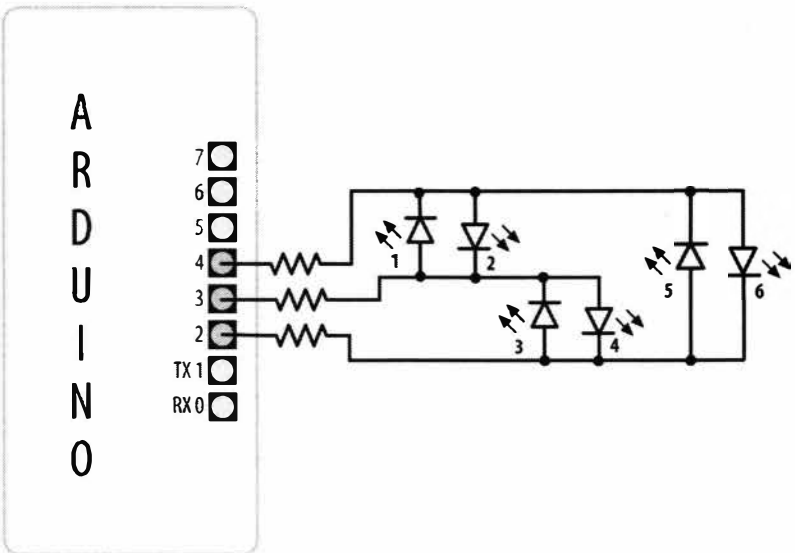


Рис. 7.11. Использование метода чарлиплексирования для управления шестью светодиодами с помощью трех контактов платы

Листинг 7.13. Управление несколькими светодиодами методом чарлиплексирования

```
/*
* Скетч Charlieplexing
* Последовательно включает и выключает три светодиода.
*/
```

```
int pins[] = {2,3,4}; // Номера контактов для управления светодиодами

// Следующие две строки кода вычисляют количество контактов
// и светодиодов на основе информации в массиве pins
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {2,1}, {1,0}, {2,0} }; // Сопоставляем
                                                           // контакты светодиодам

void setup()
{
   // Ничего не делаем
}

void loop()
{
   for(int i=0; i < NUMBER_OF_LEDS; i++)
   {
      lightLed(i); // Последовательно включаем каждый светодиод
      delay(1000);
   }
}

// Эта функция включает требуемый светодиод. Нумерация светодиодов
// начинается с 0
void lightLed(int led)
{
   // Следующие четыре строки кода преобразовывают номер светодиода
   // в номера контактов
   int indexA = pairs[led/2][0];
   int indexB = pairs[led/2][1];
   int pinA = pins[indexA];
   int pinB = pins[indexB];

   // Выключаем все контакты, не подключенные к заданному светодиоду
   for(int i=0; i < NUMBER_OF_PINS; i++)
   {
      if(pins[i] != pinA && pins[i] != pinB)
      {
         // Если этот контакт не является одним из требуемых
         pinMode(pins[i], INPUT);    // Задаем для него входной режим работы
         digitalWrite(pins[i],LOW); // Отключаем повышающий резистор
      }
   }

   // Теперь включаем контакты для требуемого светодиода
   pinMode(pinA, OUTPUT);
   pinMode(pinB, OUTPUT);
```

```
    if( led % 2 == 0)
    {
       digitalWrite(pinA,LOW);
       digitalWrite(pinB,HIGH);
       }
    else
    {
       digitalWrite(pinB,LOW);
       digitalWrite(pinA,HIGH);
    }
}
```

Обсуждение работы решения и возможных проблем

Этот метод мультиплексирования называется *чарлиплексированием* по имени Чарли Аллена (Charlie Allen) из компании Microchip Technology, Inc., который его разработал и опубликовал. Метод основан на том обстоятельстве, что светодиоды включаются только при правильном подключении: когда анод более положительный, чем катод. В табл. 7.3 приводится список комбинаций выходных сигналов на трех контактах управления и соответствующие состояния шести управляемых ими светодиодов (см. рис. 7.9). Обозначение L означает `LOW` (низкий уровень), H — `HIGH` (высокий уровень), а i — `INPUT` (входной режим работы). Контакт, находящийся во входном режиме работы, по сути, отключен от схемы.

Таблица 7.3. *Комбинации уровней контактов и соответствующие состояния светодиодов*

Контакты			Светодиоды					
4	3	2	1	2	3	4	5	6
L	L	L	0	0	0	0	0	0
L	H	i	1	0	0	0	0	0
H	L	i	0	1	0	0	0	0
i	L	H	0	0	1	0	0	0
i	H	L	0	0	0	1	0	0
L	i	H	0	0	0	0	1	0
H	i	L	0	0	0	0	0	1

Количество светодиодов можно увеличить до 12, добавив всего лишь еще один контакт управления. В таком случае первые шесть светодиодов подключаются так же, как и в предыдущем примере (см. рис. 7.11), а дополнительные шесть, как показано на рис. 7.12.

Для управления удвоенным количеством светодиодов предыдущий скетч (см. листинг 7.13) нужно модифицировать, добавив дополнительный контакт в массив pins:

```
byte pins[] = {2,3,4,5}; // Номера контактов для управления светодиодами
```

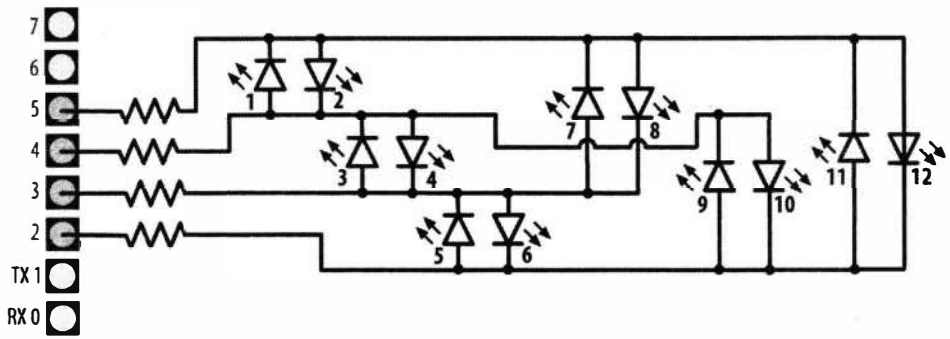


Рис. 7.12. Использование метода чарлиплексирования для управления 12 светодиодами с помощью четырех контактов платы

Также необходимо откорректировать строку кода, сопоставляющую контакты светодиодам, добавив в массив `pairs` дополнительные пары:

```
byte pairs[NUMBER_OF_LEDS/2][2] = { {0,1}, {1,2}, {0,2}, {2,3}, {1,3}, {0,3} };
```

Остальной код остается таким же, но в цикле `loop()` будут обрабатываться 12 светодиодов, т. к. их количество определяется по количеству элементов массива `pins`.

Поскольку чарлиплексирование состоит в управлении контактами Arduino таким образом, что в единицу времени включается только один светодиод, создание впечатления одновременного включения нескольких светодиодов является в этом случае более сложной задачей. Однако несколько светодиодов можно включать одновременно с помощью метода мультиплексирования, модифицированного для использования с чарлиплексированием.

В листинге 7.14 приводится код скетча, в котором этот подход используется для создания линейного индикатора, включая последовательность светодиодов, количество которых пропорционально значению сигнала датчика, подаваемого на аналоговый контакт A0.

Листинг 7.14. Создание линейного индикатора с помощью модифицированного чарлиплексирования

```
/* Скетч создания линейного индикатора с использованием чарлиплексирования
*/

byte pins[] = {2,3,4};
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {2,1}, {1,0}, {2,0} }; // Сопоставляем
                                                           // контакты светодиодам

int ledStates = 0; // Переменная для хранения состояний для 15 светодиодов
int refreshedLed;  // Светодиод, состояние которого обновляется
```

```
void setup()
{
   // Ничего не делаем
}

void loop()
{
   const int analogInPin = 0; // Входной аналоговый контакт
                              // для подключения потенциометра

   // Код из решения для создания линейного индикатора
   int sensorValue = analogRead(analogInPin); // Считываем входной
                                              // аналоговый сигнал
   // Сопоставляем полученное значение количеству светодиодов
   int ledLevel = map(sensorValue, 0, 1023, 0, NUMBER_OF_LEDS);
   for (int led = 0; led < NUMBER_OF_LEDS; led++)
   {
      if (led < ledLevel )
      {
         setState(led, HIGH); // Включаем светодиоды ниже полученного уровня
      }
      else
      {
         setState(led, LOW); // Выключаем светодиоды выше уровня
      }
   }
   ledRefresh();
}

void setState( int led, bool state)
{
   bitWrite(ledStates,led, state);
}

void ledRefresh()
{
   // При каждом вызове обновляем состояние другого светодиода
   if( refreshedLed++ > NUMBER_OF_LEDS) // Переходим к следующему светодиоду
      refreshedLed = 0; // Повторяем, начиная с первого светодиода,
                        // если все были обновлены

   if( bitRead(ledStates, refreshedLed ) == HIGH)
      lightLed( refreshedLed );

   else
      if(refreshedLed == 0) // Выключаем все светодиоды, если контакт 0 выключен
         for(int i=0; i < NUMBER_OF_PINS; i++)
         digitalWrite(pins[i],LOW);
}
```

```
// Эта функция идентична функции из скетча решения
// Она включает требуемый светодиод. Нумерация светодиодов
// начинается с 0
void lightLed(int led)
{
   // Следующие четыре строки кода преобразовывают номер светодиода
   // в номера контактов
   int indexA = pairs[led/2][0];
   int indexB = pairs[led/2][1];
   int pinA = pins[indexA];
   int pinB = pins[indexB];

   // Выключаем все контакты, не подключенные к заданному светодиоду
   for(int i=0; i < NUMBER_OF_PINS; i++)
   {
      if(pins[i] != pinA && pins[i] != pinB)
      {
         // Если этот контакт не является одним из требуемых
         pinMode(pins[i], INPUT);    // Задаем для него входной режим работы
         digitalWrite(pins[i],LOW); // Отключаем повышающий резистор
      }
   }

   // Теперь включаем контакты для требуемого светодиода
   pinMode(pinA, OUTPUT);
   pinMode(pinB, OUTPUT);
   if( led % 2 == 0)
   {
      digitalWrite(pinA,LOW);
      digitalWrite(pinB,HIGH);
   }
   else
   {
      digitalWrite(pinB,LOW);
      digitalWrite(pinA,HIGH);
   }
}
```

В скетче используются значения битов переменной `ledStates` для представления состояния светодиодов (0 — выключен, 1 — включен). Функция `refresh()` проверяет значение каждого бита и включает светодиоды для битов со значением 1. Вызов функции должен осуществляться быстро и постоянно, чтобы избежать мигания светодиодов.

Добавление пауз в код может оказывать отрицательное воздействие на эффект инерционности зрительного восприятия, благодаря которому человеческий глаз не замечает мигания светодиодов, включающихся и выключающихся с высокой частотой, а воспринимает их как постоянно включенные.

Вместо явного вызова функции `refresh()` для обновления состояния светодиодов ее можно вызывать посредством прерывания. Прерывания по таймеру подробно рассматриваются в *главе 18*, но в листинге 7.15 приводится пример одного способа использования прерывания для обновления состояния светодиодов. Для создания прерываний в скетче задействована библиотека FrequencyTimer2, устанавливаемая с помощью Менеджера библиотек (установка библиотек сторонних разработчиков подробно рассматривается в *разд. 16.2*).

Листинг 7.15. Обновление состояния светодиодов с использованием прерывания

```
#include <FrequencyTimer2.h> // Библиотека для создания прерывания

byte pins[] = {2,3,4};
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {2,1}, {1,0}, {2,0} };
int ledStates = 0; //Переменная для хранения состояний для 15 светодиодов

int refreshedLed; // Светодиод, состояние которого обновляется

void setup()
{
   FrequencyTimer2::setPeriod(20000/NUMBER_OF_LEDS); // Задаем период
   // Следующая строка кода указывает объекту FrequencyTimer2,
   // какую функцию нужно вызывать (ledRefresh)
   FrequencyTimer2::setOnOverflow(ledRefresh);
   FrequencyTimer2::enable();
}

void loop()
{
   const int analogInPin = 0; // Входной аналоговый контакт
                              // для подключения потенциометра

   // Код из решения для создания линейного индикатора
   int sensorValue = analogRead(analogInPin); // Считываем входной аналоговый сигнал
   // Сопоставляем полученное значение количеству светодиодов
   int ledLevel = map(sensorValue, 0, 1023, 0, NUMBER_OF_LEDS);
   for (int led = 0; led < NUMBER_OF_LEDS; led++)
   {
      if (led < ledLevel )
      {
         setState(led, HIGH); // Включаем светодиоды ниже полученного уровня
      }
```

```
      else
      {
         setState(led, LOW); // Выключаем светодиоды выше уровня
      }
   }
   // Состояние светодиода обновляется теперь не в цикле loop(),
   // а при обработке прерывания, создаваемого FrequencyTimer2
}
// Остальной код такой же, что и в предыдущем примере
```

Объект `FrequencyTimer2` устанавливает период прерывания величиной 1666 микросекунд (20 мс, разделенное на количество светодиодов, равное 12). Затем методу `FrequencyTimer2::setOnOverflow` указывается функция (`ledRefresh`), которую нужно вызывать при каждом активировании таймера. Библиотека FrequencyTimer2 совместима с ограниченным количеством плат: Arduino Uno (и, скорее всего, с большинством плат с микроконтроллером ATmega328), Arduino Mega и с несколькими версиями платы Teensy. Более подробная информация по этой библиотеке предлагается на веб-сайте PJRC (**https://oreil.ly/e-KTE**).

Дополнительная информация

Прерывания по таймеру более подробно рассматриваются в *главе 18*.

7.11. Управление 7-сегментным светодиодным дисплеем

ЗАДАЧА

Требуется отображать цифры на 7-сегментном цифровом светодиодном дисплее.

РЕШЕНИЕ

Скетч в листинге 7.16 отображает цифры от 0 до 9 на одноразрядном 7-сегментном светодиодном дисплее с общим анодом, подключенным к плате Arduino, как показано на рис. 7.13. В случае использования другого дисплея, его схема контактов может отличаться от указанной здесь, поэтому сверьтесь со справочным листком на конкретный дисплей. Также, если используется дисплей с общим катодом, подключите вывод катода к контакту «земли» платы Arduino. Отображение цифр на дисплее осуществляется включением соответствующих комбинаций его сегментов.

Листинг 7.16. Отображение цифр на 7-сегментном светодиодном дисплее

```
/*
* Скетч SevenSegment
* Отображает цифры от 0 до 9 на одноразрядном светодиодном дисплее
  с периодом отображения длительностью в 1 секунду
*/
```

```
// Биты, представляющие сегменты с A по G, включая десятичную точку, для цифр 0-9
const byte numeral[10] =
{
   //ABCDEFG+dp
   B11111100, // 0
   B01100000, // 1
   B11011010, // 2
   B11110010, // 3
   B01100110, // 4
   B10110110, // 5
   B00111110, // 6
   B11100000, // 7
   B11111110, // 8
   B11100110, // 9
};

// Контакты для десятичной точки (dp) и каждого сегмента
// dp,G,F,E,D,C,B,A
const int segmentPins[8] = { 5,8,9,7,6,4,3,2};

void setup()
{
   for(int i=0; i < 8; i++)
   {
      pinMode(segmentPins[i], OUTPUT); // Задаем выходной режим работы
            // для контактов управления сегментами и десятичной точкой
   }
}

void loop()
{
   for(int i=0; i <= 10; i++)
   {
      showDigit(i);
      delay(1000);
   }
   // Последнее значение счетчика i будет 10, что выключит дисплей
   delay(2000); // Выдерживаем 2-секундную паузу в выключенном
                // состоянии дисплея
}

// Отображает цифры от 0 до 9 на 7-сегментном светодиодном дисплее
// Любое значение вне пределов диапазона 0-9 выключает дисплей
void showDigit(int number)
{
   bool isBitSet;
```

```
    for(int segment = 1; segment < 8; segment++)
    {
       if( number < 0 || number > 9)
       {
          isBitSet = 0; // Выключаем все сегменты
       }
       else
       {
          // Значение переменной isBitSet будет true, если этот бит равен 1
          isBitSet = bitRead(numeral[number], segment);
       }
       isBitSet = ! isBitSet; // Удалите эту строку при использовании
                              // дисплея с общим катодом
       digitalWrite(segmentPins[segment], isBitSet);
    }
}
```

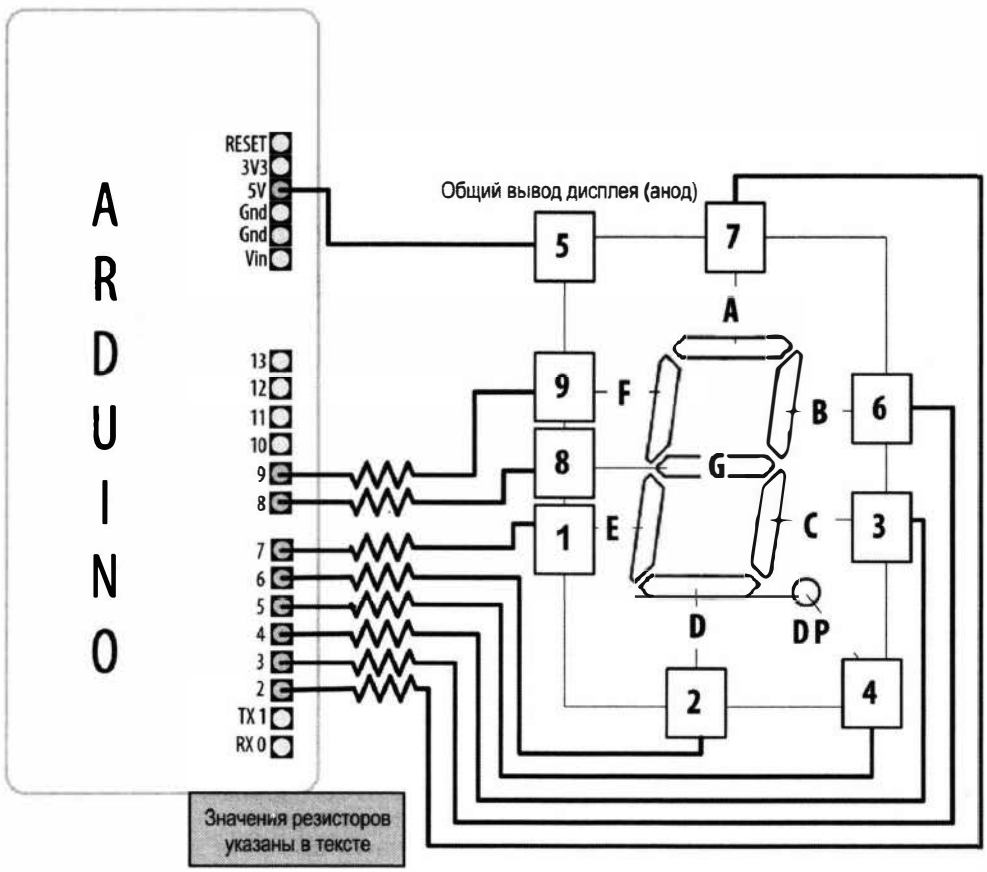


Рис. 7.13. Подключение 7-сегментного дисплея к плате Arduino

Обсуждение работы решения и возможных проблем

Идентификаторы сегментов, которые нужно включить для отображения каждой цифры, хранятся в массиве `numeral`. Каждая отдельная цифра представлена одним байтом, биты которого выводят сегменты и десятичную точку дисплея.

Номера контактов для управления сегментами хранятся в массиве `segmentPins`. Функция `showDigit()` проверяет, находится ли переданное ей число в пределах диапазона чисел от 0 до 9, и если число действительное, проверяет значение каждого его бита и включает соответствующий сегмент дисплея для битов со значением 1. Функция `bitRead()` рассматривается более подробно в *разд. 3.12*.

Как упоминалось в *разд. 7.4*, чтобы включить сегмент дисплея с общим катодом, на соответствующий контакт управления необходимо подать сигнал высокого уровня (`HIGH`), а с общим анодом — низкого (`LOW`). Код скетча предназначен для использования с дисплеями с общим анодом, поэтому в нем инвертируется значение для включения сегмента — 0 меняется на 1, а 1 — на 0:

```
isBitSet = ! isBitSet; // Удалите эту строку при использовании дисплея с общим катодом
```

Для обращения битов здесь используется унарный оператор `!`, который более подробно рассматривается в *разд. 2.20*. В случае использования дисплея с общим катодом эту строку кода следует удалить или закомментировать.

7.12. Управление многоразрядным 7-сегментным светодиодным дисплеем с использованием мультиплексирования

ЗАДАЧА

Требуется отображать числа на многоразрядном 7-сегментном светодиодном дисплее.

РЕШЕНИЕ

Для решения этой задачи обычно требуется использовать мультиплексирование. Подобно тому, как в предыдущих разделах мультиплексированные строки и столбцы светодиодной матрицы соединялись в массив, здесь сегменты для каждой цифры соединяются вместе. На рис. 7.14 показано подключение 4-разрядного 7-сегментного светодиодного дисплея к плате Arduino. Скетч для управления таким дисплеем приводится в листинге 7.17.

На рис. 7.14 показано подключение дисплея LTC-2623 компании Lite-On Electronics. При использовании другого дисплея можно задействовать те же контакты управления платы Arduino, но соответствующие контакты сегментов дисплея нужно уточнить по его справочному листку. Дисплей LTC-2623 имеет общий анод. Для использования дисплеев с общим катодом необходимо выполнить две корректировки. Первая: эмиттеры транзисторов соединить вместе и подключить на «землю», а

их коллекторы подключить к соответствующему контакту дисплея. Вторая: удалить или закомментировать эту строку кода:

```
isBitSet = ! isBitSet;
```

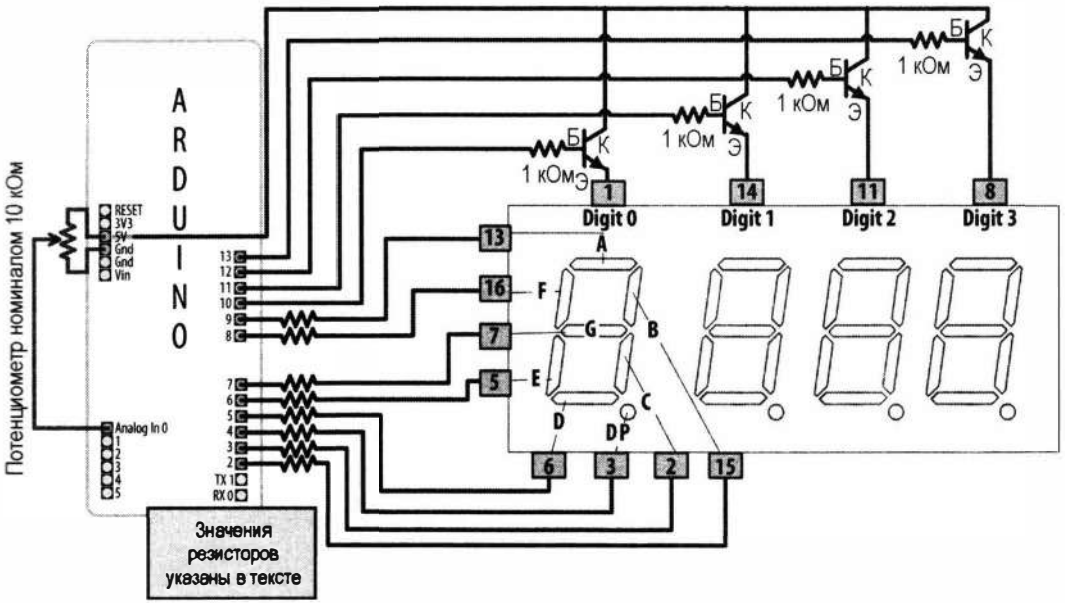


Рис. 7.14. Подключение 4-разрядного 7-сегментного дисплея (LTC-2623) к плате Arduino

Листинг 7.17. Управление 4-разрядным 7-сегментным светодиодным дисплеем

```
/*
* Скетч SevenSegmentMpx
* Отображает на 4-разрядном 7-сегментном дисплее числа
  в диапазоне от 0 до 9999
* Здесь отображается числовое значение сигнала, считываемое с датчика,
  подключенного к аналоговому контакту A0
*/

// Биты, представляющие сегменты A по G, включая десятичную точку, для цифр 0-9
const int numeral[10] =
{
   //ABCDEFG /dp
   B11111100, // 0
   B01100000, // 1
   B11011010, // 2
   B11110010, // 3
   B01100110, // 4
   B10110110, // 5
   B00111110, // 6
```

```
   B11100000, // 7
   B11111110, // 8
   B11100110, // 9
};

// Контакты для десятичной точки (dp) и каждого сегмента
// dp,G,F,E,D,C,B,A
const int segmentPins[] = { 4, 7,8,6,5,3,2,9};

const int nbrDigits= 4; // Количество разрядов дисплея
//разряды 1 2 3 4
const int digitPins[nbrDigits] = { 10,11,12,13};

void setup()
{
   for(int i=0; i < 8; i++)
   pinMode(segmentPins[i], OUTPUT); // Задаем выходной режим работы
         // для контактов управления сегментами и десятичной точкой

   for(int i=0; i < nbrDigits; i++)
   pinMode(digitPins[i], OUTPUT);
}

void loop()
{
   int value = analogRead(0);
   showNumber(value);
}

void showNumber(int number)
{
   if(number == 0)
      showDigit( 0, nbrDigits-1) ; // Отображаем 0 в самом правом разряде
   else
   {
      // Отображаем номер каждого разряда
      // Номер самого левого разряда - 0, самого правого — на 1 меньше,
      // чем количество разрядов
      for( int digit = nbrDigits-1; digit >= 0; digit--)
      {
         if(number > 0)
         {
            showDigit( number % 10, digit) ;
            number = number / 10;
         }
      }
   }
}
```

```
// Отображаем заданную цифру в заданном разряде дисплея
void showDigit( int number, int digit)
{
   digitalWrite( digitPins[digit], HIGH );
   for(int segment = 1; segment < 8; segment++)
   {
      bool isBitSet = bitRead(numeral[number], segment);
      // Значение переменной isBitSet будет true, если это бит равен 1
      isBitSet = ! isBitSet; // Удалите эту строку при использовании
                             // дисплея с общим катодом
      digitalWrite( segmentPins[segment], isBitSet);
   }
   delay(5);
   digitalWrite( digitPins[digit], LOW );
}
```

Обсуждение работы решения и возможных проблем

В скетче решения используется функция `showDigit()`, похожая на одноименную функцию из решения *разд. 7.11*. В рассматриваемом случае функции в аргументах передаются цифра для отображения и разряд, в котором ее нужно отобразить. Логика включения сегментов цифры такая же, как в функции из решения *разд. 7.11*, но здесь функция вдобавок устанавливает высокий уровень на контакте разряда, чтобы отображать только цифру этого разряда (более подробное объяснение см. в предыдущих решениях с использованием мультиплексирования).

7.13. Управление многоразрядным 7-сегментным светодиодным дисплеем с использованием минимального количества контактов

ЗАДАЧА

Требуется управлять многоразрядным 7-сегментным светодиодным дисплеем, используя минимальное количество контактов платы Arduino.

РЕШЕНИЕ

В этом решении для управления 4-разрядным дисплеем с общим катодом применена адаптерная плата с микросхемой драйвера дисплея HT16K33 — например, KW4-56NXBA-P компании LuckyLight или BL-Q56C-43 компании Betlux. Применение драйвера дисплея HT16K33 — это лучшее решение, чем решение с использованием мультиплексирования, рассмотренное в *разд 7.12*, поскольку мультиплексирование и декодирование цифр осуществляются аппаратно. Драйвер дисплея HT16K33 на адаптерной плате предлагается широким кругом поставщиков. В частности,

в ассортименте компании Adafruit имеется адаптерная плата LED Matrix Backpack (артикул 877), предназначенная для совместной работы с ее же 4-разрядным 7-сегментным дисплеем, выпускает она также сборки этих драйверов в комплекте с 7-сегментными дисплеями разных цветов (**https://www.adafruit.com/category/103**).

Подключение драйвера дисплея к плате Arduino показано на рис. 7.15, а в листинге 7.18 приводится скетч, который отображает на дисплее числа в диапазоне от 0 до 9999.

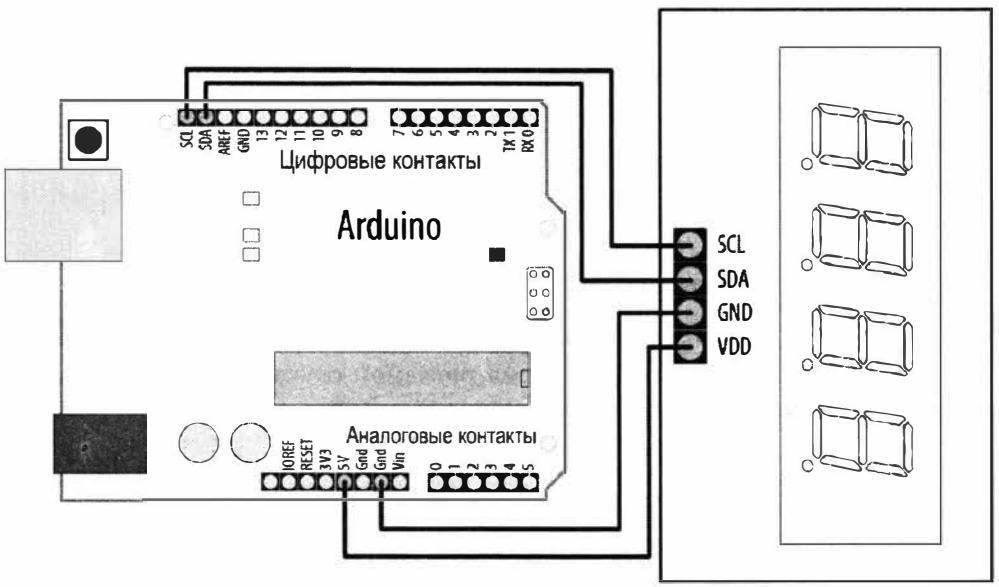


Рис. 7.15. Подключение 4-разрядного 7-сегментного дисплея с драйвером HT16K33 к плате Arduino

Листинг 7.18. Скетч для работы с драйвером дисплея HT16K33

```
/*
Скетч HT16K33 seven segment display
*/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include "Adafruit_LEDBackpack.h"

Adafruit_7segment matrix = Adafruit_7segment();

const int numberOfDigits = 4; // Замените 4 на используемое количество разрядов
const int maxCount = 9999;

int number = 0;
```

```
void setup()
{
   Serial.begin(9600);
   matrix.begin(0x70);      // Инициализируем объект дисплея
   matrix.println(number); // Посылаем число (изначально 0) на дисплей
   matrix.writeDisplay();  // Обновляем состояние дисплея
}

void loop()
{
   // Отображаем число, полученное с последовательного порта,
   // завершающееся символом конца строки
   if (Serial.available())
   {
      char ch = Serial.read();
      if ( ch == '\n')
      {
         if (number <= maxCount)
         {
            matrix.println(number);
            matrix.writeDisplay();
            number = 0; // Обнуляем число
         }
      }
      else
      number = (number * 10) + ch - '0'; // Подробнее об этом см. в главе 4
   }
}
```

Обсуждение работы решения и возможных проблем

В этом решении взаимодействие платы Arduino с драйвером дисплея HT16K33 осуществляется посредством интерфейса I^2C. Библиотека Adafruit_LEDBackpack предоставляет интерфейс для аппаратной части драйвера через экземпляр `matrix` объекта `Adafruit_7segment` (интерфейс I^2C подробно рассматривается в *главе 13*).

Скетч принимает четырехзначные числа по последовательному порту и отображает их на дисплее (код для работы с последовательным портом из раздела скетча `loop` подробно описан в *главе 4*). Функция `matrix.println()` отправляет полученные числа на драйвер HT16K33, а функция `matrix.writeDisplay()` обновляет дисплей последним отправленным на него значением.

Примененная в этом решении коммутационная плата (драйвер) работает с четырехзначным 7-сегментным дисплеем, но если вы приобретете более универсальную коммутационную плату HT16K33 (например, артикул Adafruit 1427), то сможете использовать ее с одно- или двузначными дисплеями для отображения до восьми цифр. При создании многоразрядных светодиодных дисплеев из одноразрядных элементов соответствующие сегменты всех элементов необходимо соединить вме-

сте. Конкретную информацию по подключению сегментов дисплея необходимо смотреть в его справочном листке, а также в справочном листке используемой адаптерной платы драйвера дисплея HT16K33.

7.14. Управление светодиодной матрицей с использованием драйвера дисплея MAX72*xx*

ЗАДАЧА

Требуется управлять светодиодной матрицей 8×8, используя минимальное количество контактов платы Arduino.

РЕШЕНИЕ

Подобно решению из *разд. 7.13*, количество контактов для управления светодиодной матрицей можно уменьшить, применив микросхему драйвера дисплея. В этом решении используется популярная микросхема драйвера светодиодного дисплея MAX7219 или MAX7221. Подключение драйвера дисплея к плате Arduino и светодиодной матрицы к драйверу показано на рис. 7.16, а в листинге 7.19 приводится скетч для работы с этой схемой.

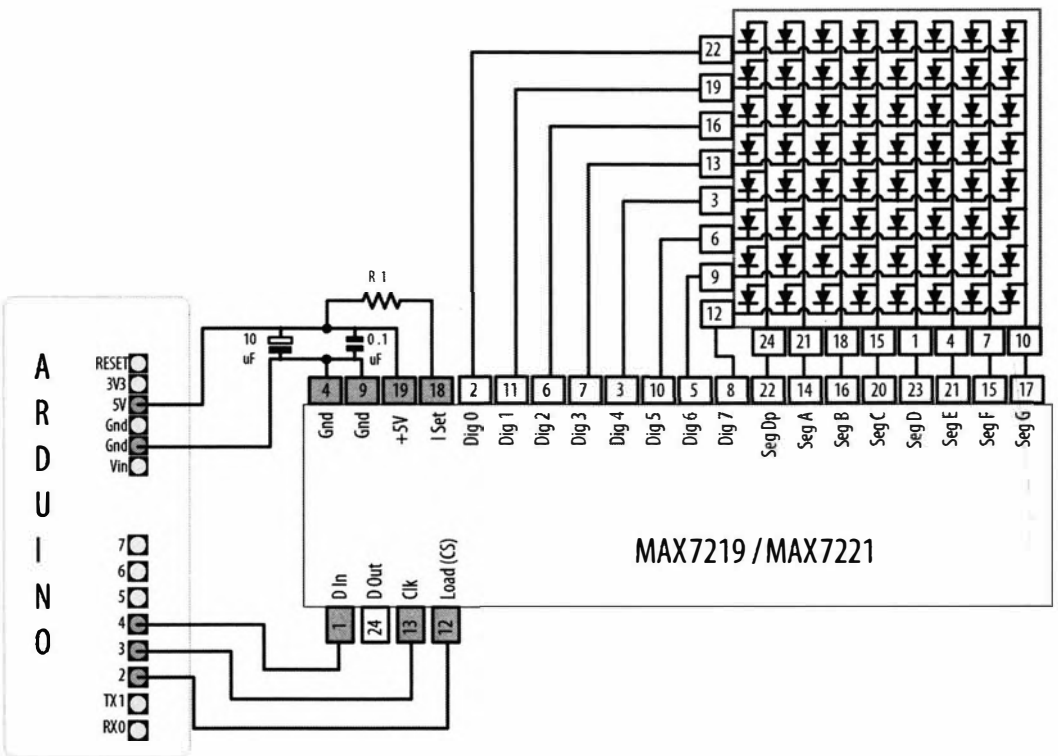


Рис. 7.16. Подключение драйвера дисплея MAX72*xx* к плате Arduino для управления светодиодной матрицей 8×8

В скетче используется библиотека с обширными возможностями MD_MAX72XX, которая поддерживает отображение на дисплее текста и фигур, а также выполнение различных трансформаций отображаемого содержимого. Библиотека MD_MAX72XX устанавливается с помощью Менеджера библиотек среды Arduino IDE (подробная информация по установке библиотек приведена в *разд. 16.2*).

Листинг 7.19. Управление светодиодной матрицей с помощью драйвера дисплея MAX72xx

```
/*
Скетч 7219 Matrix demo
*/

#include <MD_MAX72xx.h>

// Контакты платы Arduino для управления драйвером 7219
#define LOAD_PIN 2
#define CLK_PIN 3
#define DATA_PIN 4

// Конфигурируем устройство
#define MAX_DEVICES 1
#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW
MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, LOAD_PIN, MAX_DEVICES);

void setup()
{
   mx.begin();
}

void loop()
{
   mx.clear(); // Очищаем дисплей
   // Отображаем строки и столбцы
   for (int r = 0; r < 8; r++)
   {
      for (int c = 0; c < 8; c++)
      {
         mx.setPoint(r, c, true); // Выключаем каждый светодиод модуля
         delay(50);
      }

      // Проходим в цикле по всем возможным уровням яркости
      for (int k = 0; k <= MAX_INTENSITY; k++)
      {
         mx.control(MD_MAX72XX::INTENSITY, k);
```

```
            delay(100);
        }
    }
}
```

Обсуждение работы решения и возможных проблем

В начале скетча создается экземпляр `mx` объекта `MD_MAX72XX`, которому в параметрах передается тип устройства, номера контактов для данных загрузки и сигнала тактирования, а также максимальное количество устройств (при последовательном соединении модулей). В главном цикле `loop()` выполняется очистка дисплея, а затем с помощью функции `setPoint()` включаются пикселы (светодиоды) матрицы. Включив строку матрицы, скетч проходит в цикле через все возможные уровни яркости, а затем переходит к обработке следующей строки.

В скетче указаны номера контактов для зеленых светодиодов двухцветной светодиодной матрицы 8×8 компании Adafruit (артикул 458). При использовании другой светодиодной матрицы обратитесь к справочному листку на нее, чтобы определить контакты для ее строк и столбцов. Скетч также будет работать и с одноцветной матрицей, поскольку он использует только один из двух цветов матрицы. Если обнаружится, что матрица отображает текст в обратном направлении или не в ожидаемой ориентации, можно попробовать исправить эту ошибку, изменив тип устройства в строке:

```
#define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW с PAROLA_HW
```

на `GENERIC_HW`, `ICSTATION_HW` или `FC16_HW`. В примерах библиотеки MD_MAX72XX содержится тестовый скетч MD_MAX72xx_HW_Mapper, который выполняет тестирование и помогает определить, какой тип устройства использовать.

Резистор R1 на рис. 7.16 ограничивает максимальный ток, который может протекать через светодиод. В табл. 7.4 приведены данные номиналов токоограничивающих резисторов из справочного листка драйвера MAX72*xx* для нескольких значений прямого напряжения светодиода и величины протекающего через него тока.

***Таблица 7.4.** Номиналы токоограничивающих резисторов (из справочного листка драйвера MAX72xx)*

Ток	Прямое напряжение светодиода				
	1,5 В	2,0 В	2,5 В	3,0 В	3,5 В
40 мА	12 кОм	12 кОм	11 кОм	10 кОм	10кОм
30 мА	18 кОм	17 кОм	16 кОм	15 кОм	14 кОм
20 мА	30 кОм	28 кОм	26 кОм	24 кОм	22 кОм
10 мА	68 кОм	64 кОм	60 кОм	56 кОм	51 кОм

Величина прямого напряжения зеленых светодиодов матрицы, представленной на рис. 7.16, составляет 2 вольта, а прямой ток — 20 мА. Согласно данным табл. 7.3,

для этой комбинации напряжения и тока требуется резистор сопротивлением 38 кОм, но для надежности лучше использовать резистор номиналом 30 или 33 кОм. Конденсаторы емкостью 0,1 и 10 мкФ, подключенные параллельно линиям питания, требуются для того, чтобы предотвратить возникновение импульсных помех при включении и выключении светодиодов матрицы (дополнительная информация по таким конденсаторам приводится в *разд. «Использование развязывающих конденсаторов» приложения 3*).

Дополнительная информация

Подробная информация по микросхеме драйвера дисплея MAX72*xx* приведена в ее справочном листке (**https://oreil.ly/IH7U7**).

7.15. Увеличение количества выходных аналоговых контактов платы с использованием микросхемы расширения ШИМ-сигнала

ЗАДАЧА

Требуется управлять яркостью большего количества светодиодов, чем способна поддерживать плата Arduino.

РЕШЕНИЕ

Микросхема драйвера ШИМ-сигнала PCA9685 позволяет обеспечить управление до 16 светодиодов, используя всего лишь два контакта интерфейса I^2C. Компания Adafruit выпускает адаптерную плату с этой микросхемой (артикул 815), которая может управлять несколькими сервомашинками или светодиодами. Подключение такой платы и светодиодов к плате Arduino показано на рис. 7.17, а в листинге 7.20 приводится скетч для работы с этой схемой. В скетче используется библиотека Adafruit_PWMServoDriver, которая устанавливается с помощью Менеджера библиотек среды Arduino IDE (подробная информация по установке библиотек приведена в *разд. 16.2*).

Листинг 7.20. Создание эффекта бегущей волны управлением яркостью светодиодов с помощью микросхемы ШИМ-драйвера PCA9685

```
/*
Скетч PCA9685
Создает эффект бегущей волны, управляя 16 светодиодами, подключенными
к драйверу PCA9685
*/

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
```

```
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

void setup()
{
   pwm.begin(); // Инициализируем плату ШИМ
}

int channel = 0;
int channel_direction = 1;
int intensity = 4095;    // Максимальная яркость
int dim = intensity / 4; // Минимальная яркость

void loop()
{
   channel += channel_direction; // Инкрементируем или декрементируем номер канала

   // Выключаем все контакты
   for (int pin = 0; pin < 16; pin++)
   {
      pwm.setPin(pin, 0);
   }

   // Если канал 0, устанавливаем направление 1
   if (channel == 0)
   {
      channel_direction = 1;
   }
   else
   {
      // Если канал 1 или выше, задаем его предыдущему соседу минимальную яркость
      pwm.setPin(channel - 1, dim);
   }

   // Задаем этому каналу максимальную яркость
   pwm.setPin(channel, intensity);

   if (channel < 16)
   {
      // Если ниже канала 16, задаем следующему каналу минимальную яркость
      pwm.setPin(channel + 1, dim);
   }
   else
   {
      // Если канал 16, устанавливаем направление -1
      channel_direction = -1;
   }
   delay(75);
}
```

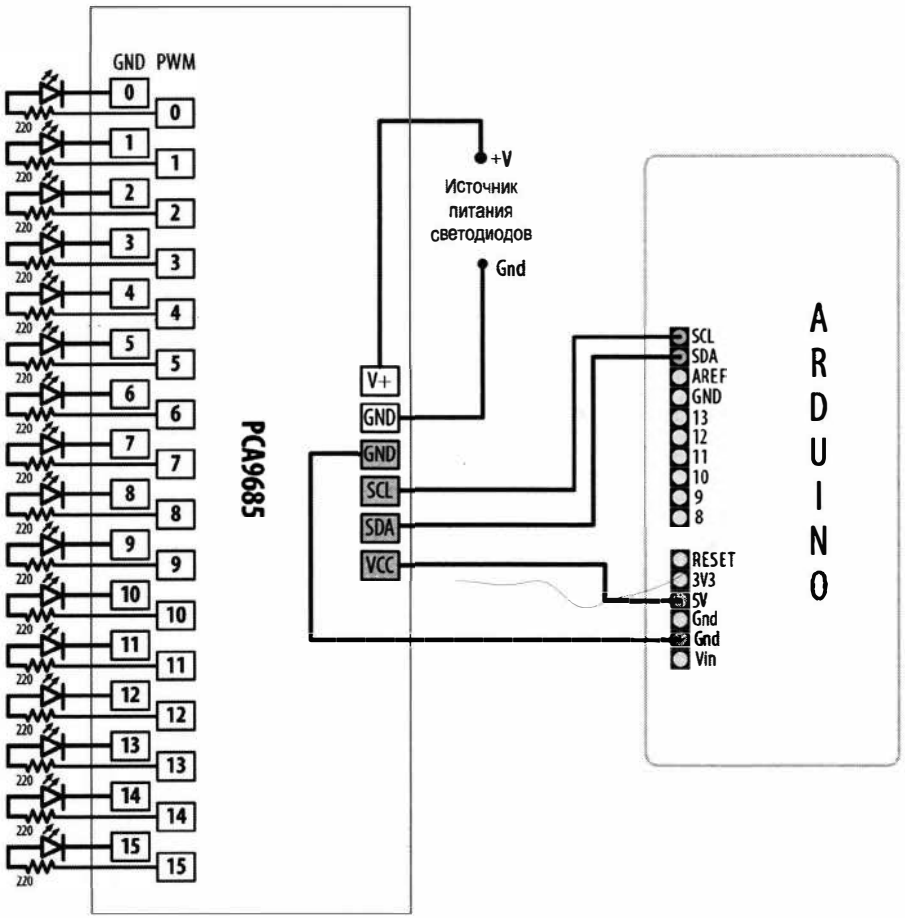


Рис. 7.17. Подключение драйвера ШИМ и светодиодов к плате Arduino

Обсуждение работы решения и возможных проблем

Скетч обрабатывает в цикле каждый канал (светодиод), устанавливая минимальную яркость для предыдущего светодиода, максимальную — для текущего и минимальную — для следующего. Светодиоды управляются несколькими основными методами. В скетче предполагается, что для драйвера PCA9685 задан I^2C адрес по умолчанию `0x40`.

Прежде чем предпринимать какие-либо действия, в скетче выполняется инициализация драйвера методом `Adafruit_PWMServoDriver.begin()`. Метод `pwm.setPin()` задает коэффициент заполнения ШИМ-сигнала для заданного канала как количество импульсов в диапазоне от 0 до 4095. В первом параметре методу передается номер канала, а во втором — яркость. Каждый цикл сигнала широтно-импульсной модуляции разбит на 4096 квантов. Значение параметра яркости обозначает количество квантов, в течение которых светодиод должен быть включен. Частоту ШИМ можно изменить с помощью метода `pwm.setPWMFreq()`, передав ему частоту в герцах от 40 до 1600.

Дополнительная информация

На веб-странице GitHub библиотеки Adafruit_PWMServoDriver (**https://oreil.ly/KRqyU**) содержится информация и о ее других функциях.

Несколько адаптерных плат драйвера можно соединить последовательно. При этом каждая плата должна иметь однозначный адрес, который ей присваивается пропаиванием контактных площадок, обозначенных с A0 по A5. Соединить цепочкой можно до 62 адаптерных плат драйвера (**https://oreil.ly/9VnOv**).

7.16. Использование в качестве дисплея аналогового измерительного прибора

ЗАДАЧА

Требуется управлять с помощью скетча стрелкой аналогового измерительного прибора. Колеблющиеся показания легче воспринимать, наблюдая за ними на аналоговом измерительном приборе. Кроме этого, такие приборы придают проектам модный вид старых устройств.

РЕШЕНИЕ

Аналоговый измерительный прибор подключается своим минусом к контакту «земли» платы Arduino через последовательный резистор (номиналом 5 кОм для типичного прибора на 1 мА), а плюсом — к контакту ШИМ платы (рис. 7.18).

Скетч для работы с этой схемой приводится в листинге 7.21. Перемещения стрелки соответствуют положению вала потенциометра.

Листинг 7.21. Аналоговый измерительный прибор в качестве дисплея

```
/*
* Скетч AnalogMeter
* Отображает выходной ШИМ-сигнал на аналоговом измерительном приборе
* Уровень выходного ШИМ-сигнала управляется потенциометром,
  подключенным к входному аналоговому контакту
*/

const int analogInPin = A0;    // Входной аналоговый контакт
                               // для подключения потенциометра
const int analogMeterPin = 9; // Выходной аналоговый контакт
                               // для подключения измерительного прибора

int sensorValue = 0; // Считываемое с потенциометра значение
int outputValue = 0; // Значение выходного ШИМ-сигнала
```

```
void setup()
{
   // Ничего не делаем
}

void loop()
{
   sensorValue = analogRead(analogInPin);     // Считываем входной аналоговый сигнал
   outputValue = map(sensorValue, 0, 1023, 0, 255); // Масштабируем его
                                    // в диапазон значений выходного аналогового сигнала
   analogWrite(analogMeterPin, outputValue);  // Подаем выходное значение на контакт
}
```

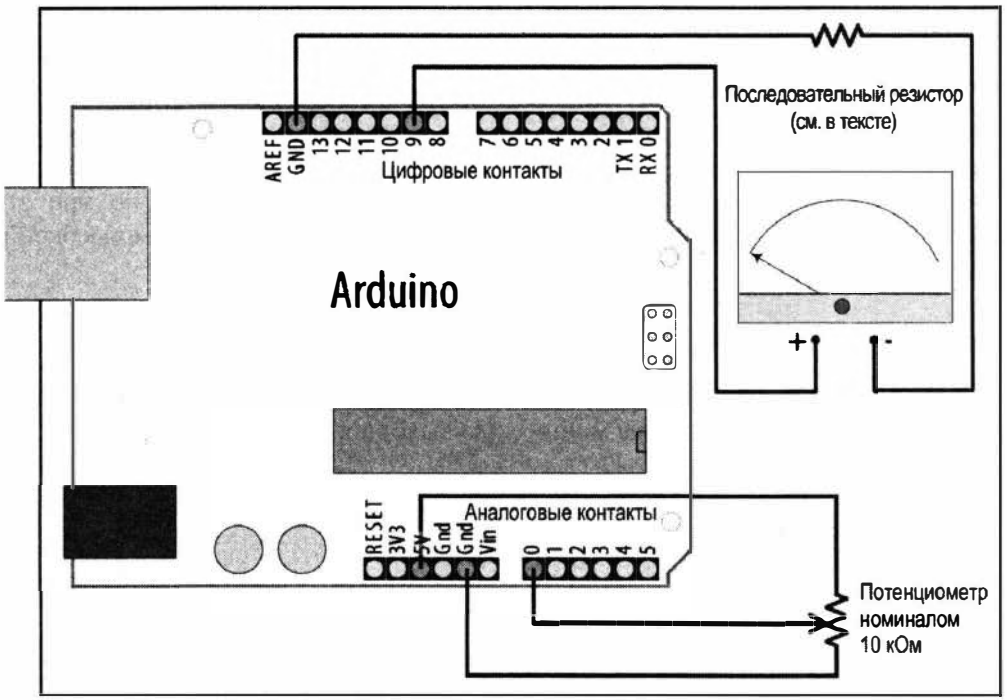


Рис. 7.18. Аналоговый измерительный прибор в качестве дисплея

Обсуждение работы решения и возможных проблем

Это версия решения из *разд. 7.2*, в которой вместо управления яркостью светодиода осуществляется управление отклонением стрелки аналогового измерительного прибора. Измерительные приборы обычно намного более чувствительные, чем светодиоды, поэтому прибор нужно подключать к плате Arduino через последовательный резистор, чтобы ограничить ток до приемлемого уровня.

Значение сопротивления такого резистора зависит от чувствительности прибора. Для прибора на 1мА резистор номиналом 5 кОм позволит получить отклонение

стрелки на всю шкалу. Допустимо использовать резистор номиналом 4,7 кОм, поскольку этот номинал легче найти, чем 5 кОм, но в таком случае максимальное значение, передаваемое функции `analogWrite()`, нужно будет уменьшить до 240 или около этого. Далее приводится соответствующая строка кода:

```
outputValue = map(sensorValue, 0, 1023, 0, 240); // Масштабируем
      // выходное значение под диапазон измерительного прибора
```

Для прибора с чувствительностью иной, чем 1 мА, надо использовать другое значение токоограничивающего резистора. Это значение в омах рассчитывается по следующей формуле:

```
Сопротивление = 5,000 / мА
```

Таким образом, для прибора с чувствительностью в 500 микроампер (0,5 мА) требуется резистор сопротивлением: 5000/0,5 = 10 000 Ом, или 10 кОм. Точно так же, для прибора с чувствительностью 10 мА требуется сопротивление величиной 500 Ом, а для 20 мА — 250 Ом.

Некоторые представленные на рынке бывшие в употреблении приборы уже оснащены встроенным последовательным резистором. В таком случае вам придется поэкспериментировать, чтобы определить требуемое сопротивление, но при этом нужно соблюдать осторожность, чтобы не подать на прибор слишком высокое напряжение, которое может вывести его из строя.

Дополнительная информация

Обратите внимание, что в *разд. 7.2* рассматривается пример использования функции `analogWrite()` для управления мощностью выходного ШИМ-сигнала.

ГЛАВА 8

Управление электродвигателями

8.0. Введение

Платформа Arduino позволяет придать объектам мобильность путем управления их электродвигателями. Для разных приложений требуются разные типы электродвигателей, и в этой главе мы рассмотрим, как с помощью Arduino управлять электродвигателями многих разных типов. В частности, мы увидим, как работать с сервомашинками, которые представляют собой электродвигатель, оснащенный встроенной схемой, позволяющей вращать вал электродвигателя на определенный угол или с определенной скоростью. Мы также узнаем о щеточных и бесщеточных электродвигателях — эти оба типа электродвигателей могут обеспечивать вращение в разных направлениях и с разными скоростями, но полагаются для этого на разные конструктивные особенности. Кроме того, в главе приводятся решения с использованием шаговых двигателей, вал которых можно поворачивать на заданное количество шагов в том или ином направлении. Наконец, вдобавок к электродвигателям, создающим вращательное движение, здесь также рассматриваются примеры по работе с реле и соленоидами.

Сервомашинки

Сервомашинки (сервоприводы) позволяют точно управлять физическим движением объекта, на который установлены, поскольку они обычно вращают свой вал не беспрерывно, а поворачивают его на определенный угол. Они идеально подходят для поворота объекта в любое положение в диапазоне от 0 до 180 градусов. Благодаря встроенному электродвигателю сервомашинки легко подключаются к плате и управляются ею.

Электродвигатель сервомашинки через набор шестерен передает вращательное движение на выходной вал, который, в свою очередь, вращает насаженную на него качалку. Выходной вал также вращает вал потенциометра, выходной сигнал которого предоставляет схеме управления сервомашинки обратную связь о положении вала. Устройство сервомашинки показано на рис. 8.1.

Стандартную сервомашинку можно модифицировать для непрерывного вращения, удалив из нее механизм обратной связи. Это позволит задавать вращение вала в одну или другую сторону, управляя скоростью его вращения с некоторой степенью точности. Эта функция немного похожа на функцию щеточных электродвигателей, рассматриваемых в *разд. 8.9*, с тем исключением, что для управления сервомашин-

ками непрерывного вращения используется библиотека Servo, а не функция `analogWrite()`, и им не требуется шилд для управления электродвигателем, поскольку драйвер электродвигателя уже встроен в сервомашинку.

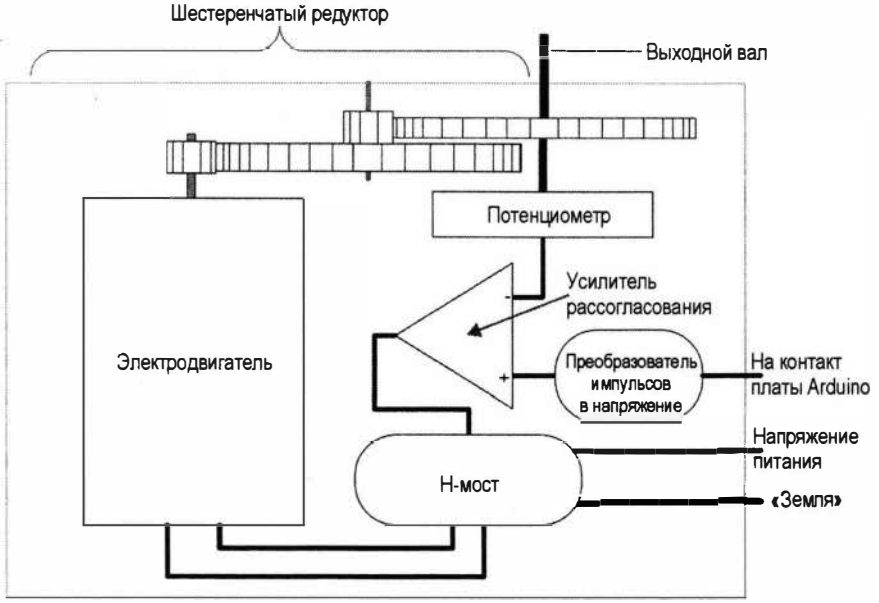


Рис. 8.1. Устройство любительской сервомашинки

К недостаткам сервомашинок непрерывного вращения можно отнести ограниченные диапазоны скорости и мощности по сравнению с обычными электродвигателями. Кроме того, точность управления их скоростью обычно хуже, чем обеспечивается шилдом драйвера электродвигателя, поскольку электронная схема управления сервомашинок предназначена для точного позиционирования вала, а не для управления скоростью его вращения. Дополнительная информация по работе с сервомашинками непрерывного вращения приводится в *разд. 8.3*.

Сервомашинки управляются изменением длительности импульса управляющего сигнала. Короткий импульс — длительностью 1 мс или меньше — заставит вал сервомашинки повернуться в одно крайнее положение, а длинные импульсы длительностью 2 мс — в другое. Импульсы длительностью между этими двумя значениями вызывают поворот вала сервомашинки на угол, пропорциональный длительности импульса (рис. 8.2). Стандарта, который бы определял точную связь длительности импульсов и угла поворота вала сервомашинки, не существует, поэтому может понадобиться поэкспериментировать с командами в скетче, чтобы подогнать ее к рабочему диапазону имеющейся сервомашинки.

Для управления сервомашинками требуются импульсы иного качества, чем предоставляемые выходным ШИМ-сигналом, создаваемым функцией `analogWrite()`. Подача на сервомашинку такого выходного ШИМ-сигнала может вывести ее из строя. Для создания управляющих сигналов для сервомашинок необходимо использовать библиотеку Servo.

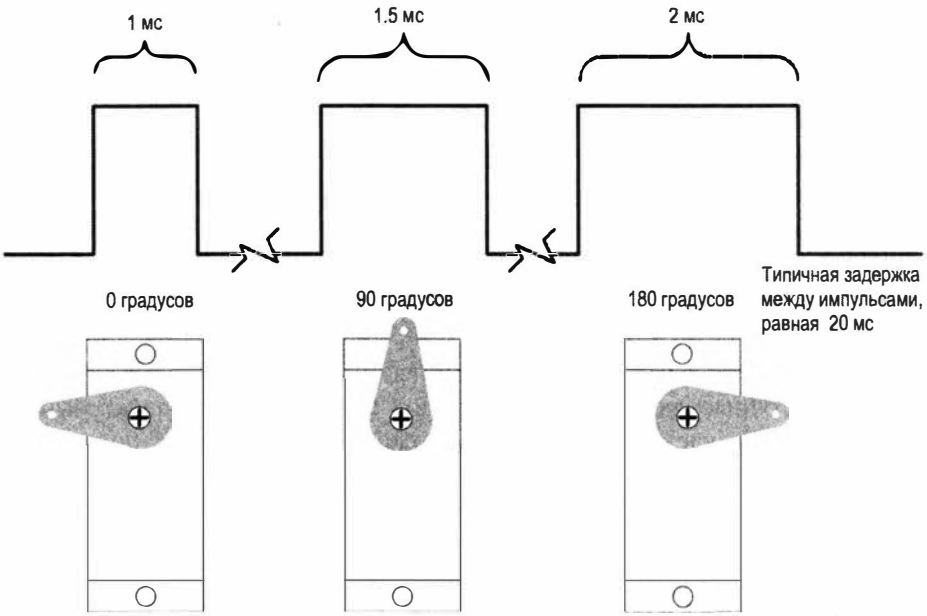


Рис. 8.2. Взаимосвязь между длительностью импульсов и углом поворота вала сервомашинки. При увеличении длительности импульсов от 1 до 2 мс вал сервомашинки поворачивается на угол, пропорциональный длительности импульсов

Соленоиды и реле

Тогда как большинство электродвигателей при подаче на них питания создают вращательное движение, *соленоиды* создают линейное поступательное движение. Соленоид состоит из катушки и расположенного внутри катушки металлического сердечника (якоря), который перемещается под воздействием магнитного поля, создаваемого при протекании тока через катушку.

Электромеханическое реле представляет собой соленоид, якорь которого замыкает или размыкает электрические контакты, т. е. это соленоид, управляющий выключателем. Реле управляются точно так же, как и соленоиды. Подобно большинству электродвигателей, реле и соленоиды потребляют больший ток, чем может быть предоставлен контактом платы Arduino, поэтому для управления ими задействуются транзисторы. В решениях этой главы показано, как использовать транзистор или внешнюю схему для управления этими устройствами.

Щеточные и бесщеточные электродвигатели

Большинство недорогих электродвигателей постоянного тока представляют собой простые устройства, в которых подача питания на обмотки *статора*, создающие магнитное поле, вызывающее вращение *ротора*, осуществляется *коллектором*, питание на который подается через графитные *щетки* (контакты). Направление вращения таких электродвигателей можно изменить на обратное, поменяв на обратную полярность питания. На рынке предлагаются электродвигатели постоянного тока

разных размеров, но для управления даже самым маленьким из них (например, электродвигателем вибратора мобильного телефона) требуется использовать транзистор или другую внешнюю схему, обеспечивающую достаточный ток. В решениях этой главы показывается, как управлять электродвигателями, используя транзистор или внешнюю схему, называемую *H-мостом*.

Основной характеристикой, берущейся в расчет при выборе электродвигателя, является его *крутящий момент*. Крутящий момент определяет величину силы вращения вала электродвигателя. Как правило, электродвигатели с более высоким крутящим моментом имеют большие размеры и больший вес, а также потребляют больший ток, чем электродвигатели с более низким крутящим моментом.

Бесщеточные электродвигатели более мощные и эффективные при таком же размере, что и щеточные двигатели, но требуют более сложных электронных схем управления. Для управления бесщеточными электродвигателями с помощью платы Arduino можно применять электронные контроллеры, используемые в радиоуправляемых моделях, которые по большей части управляются так же, как и сервомашинки.

Шаговые двигатели

Шаговые электродвигатели вращают вал на определенное количество градусов в ответ на импульсы управления. Количество градусов на каждый шаг зависит от конкретного двигателя и может находиться в диапазоне от одного или двух градусов на шаг до 30 градусов на шаг или даже более.

С платформой Arduino широко используются два типа шаговых двигателей: биполярные (обычно с четырьмя выводами двух катушек) и униполярные (с пятью или шестью выводами двух катушек). Дополнительные выводы униполярных шаговых двигателей подключены к середине обмоток. В случае пяти выводов середины обмоток соединены внутри и вывод идет от этой общей точки. Все эти подключения показаны в схемах, сопровождающих решения задач по биполярным и униполярным шаговым двигателям.

Наиболее распространенная проблема при подключении устройств, требующих подачи на них внешнего питания, возникает вследствие недосмотра по соединению вместе шин «земли» всех устройств. Шина «земли» платы Arduino должна быть подключена к шине «земли» внешнего источника питания, а также к шинами «земли» любых других внешних запитываемых устройств.

8.1. Управление угловой позицией объекта с помощью сервомашинки

ЗАДАЧА

Требуется управлять вращением объекта на угол, рассчитываемый в скетче. Например, повернуть датчик или руку робота по дуге сектора или установить ее в требуемую позицию.

РЕШЕНИЕ

Подключите сервомашинку к подходящему источнику питания. Обычно одну любительскую сервомашинку (наподобие используемых в радиоуправляемых моделях самолетов и т. п.) можно запитать от контакта выходного питания 5V платы Arduino. Вывод управления сервомашинки можно подключить к любому цифровому контакту платы Arduino.

Подключение сервомашинки к плате Arduino показано на рис. 8.3, а в листинге 8.1 приводится скетч для работы с ней. Это скетч примера Sweep, входящего в состав среды Arduino IDE. Для работы скетча нужно установить библиотеку Servo среды Arduino IDE.

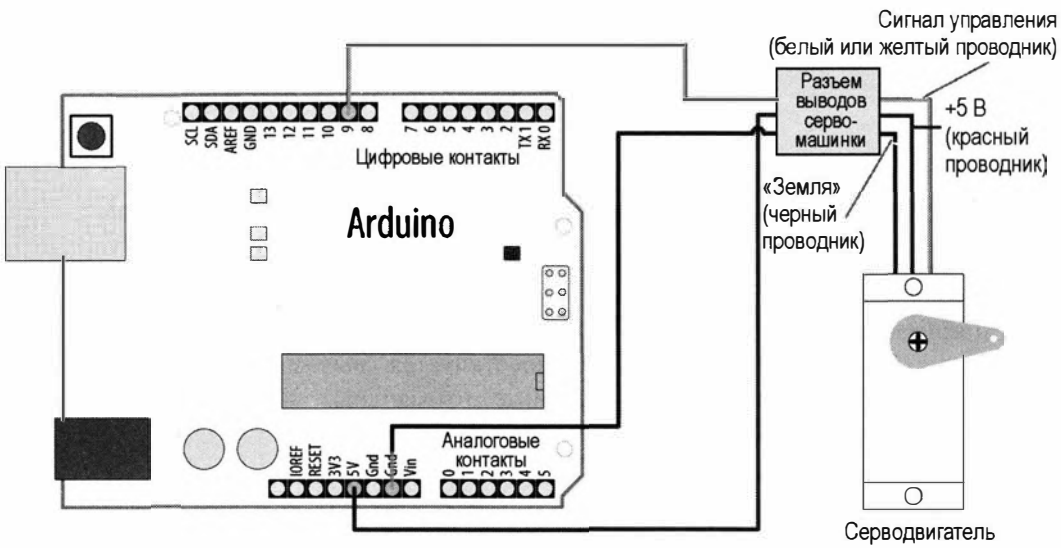


Рис. 8.3. Подключение сервомашинки для тестирования ее работы со скетчем Sweep

Листинг 8.1. Управление угловой позицией объекта с помощью сервомашинки

```
/*
* Скетч Servo rotation
*/

#include <Servo.h>

Servo myservo; // Создаем экземпляр объекта Servo для управления сервомашинкой

int angle = 0; // Переменная для хранения позиции сервомашинки

void setup()
{
   myservo.attach(9); // Подключаем к объекту Servo сервомашинку,
                      // подключенную к контакту 9 платы Arduino
}
```

```
void loop()
{
   for(angle = 0; angle < 180; angle += 1) // Делаем поворот
                            // от 0 градусов до 180 градусов
   {
      // in steps of 1 degree
      myservo.write(angle); // Проворачиваем сервомашинку до позиции,
                            // указанной в параметре angle
      delay(20);            // Пауза в 20 мс между командами управления сервомашинкой
   }
   for(angle = 180; angle >= 1; angle -= 1) // Делаем поворот
                             // от 180 градусов до 0 градусов
   {
      myservo.write(angle); // Поворачиваем сервомашинку в обратном направлении
      delay(20);           // Пауза в 20 мс между командами управления сервомашинкой
   }
}
```

Обсуждение работы решения и возможных проблем

Этот пример проворачивает вал сервомашинки от начального положения 0 градусов до конечного положения 180 градусов. Возможно, вам придется откорректировать минимальную и максимальную позиции, чтобы получить требуемый угол перемещения. Такая корректировка осуществляется вызовом метода (функции) `Servo.attach()` с передачей ему в параметрах оптимальных значений для минимального и максимального положений:

```
myservo.attach(9,1000,2000); // Вывод сигнала управления сервомашинки
// подключен к контакту 9 платы Arduino.
// Задаем параметру min значение 1000 мкс, параметру max – 2000 мкс.
```

Поскольку типичные сервомашинки реагируют на длительность импульсов в микросекундах, а не на углы, в аргументах `min` и `max`, следующих после аргумента `pin`, указывается, сколько микросекунд соответствует углам 0 и 180 градусов. Не все сервомашинки имеют рабочий диапазон перемещений от 0 до 180 градусов, поэтому, возможно, вам придется экспериментировать с конкретной используемой сервомашинкой, чтобы узнать ее рабочий диапазон перемещений.

Далее приводится объяснение параметров, передаваемых методу `servo.attach(`*pin, min, max*`)`:

- *pin*

  Номер контакта платы Arduino, к которому подключен проводник сигнала управления сервомашинки (можно использовать любой цифровой контакт).

- *min (необязательный)*

  Длительность импульсов в микросекундах, соответствующих минимальному (0 градусов) углу поворота вала сервомашинки (по умолчанию используется значение 544).

- `max (необязательный)`

  Длительность импульсов в микросекундах, соответствующих максимальному (180 градусов) углу поворота вала сервомашинки (по умолчанию используется значение 2400).

Библиотека Servo поддерживает управление до 12 сервомашинок для большинства плат Arduino и 48 — платой Arduino Mega. На платах Arduino Uno и других платах с микроконтроллером ATmega328 работа с библиотекой Servo делает невозможным использование функции analogWrite() (вывод ШИМ-сигнала) с контактами 9 и 10, даже если эти контакты не используются для подключения к ним сервомашинок. Это ограничение не относится к плате Arduino Mega и, скорее всего, к некоторым другим 32-разрядным платам. Дополнительная информация по работе с библиотекой Servo приводится на ее веб-странице справки сайта Arduino (**https://oreil.ly/_3elx**).

Требования по питанию зависят от конкретной сервомашинки и величины крутящего момента, требуемого для вращения вала.

Для питания нескольких сервомашинок может потребоваться внешний источник питания напряжением 5 или 6 вольт. В случае питания от батарей для этой цели подойдут четыре батарейки типоразмера AA. Не забывайте, что при использовании отдельного источника питания необходимо соединить его шину «земли» с шиной «земли» источника питания платы Arduino.

8.2. Управление сервомашинкой с помощью потенциометра или другого датчика

ЗАДАЧА

Требуется управлять направлением вращения и позицией сервомашинки, используя выходной сигнал потенциометра. Например, нужно организовать управление наклоном и поворотом камеры видеонаблюдения или какого-либо иного датчика. Это решение будет работать с любым сигналом переменного напряжения с датчика, которое можно считывать с контакта аналогового ввода.

РЕШЕНИЕ

Для этого решения сервомашинка подключается к плате Arduino так же, как и для предыдущего решения, с тем исключением, что к плате Arduino дополнительно подключен потенциометр, управляющий сервомашинкой (рис. 8.4). В скетче решения (листинг 8.2) используется та же библиотека Servo, что и в скетче решения в *разд. 8.1* (см. листинг 8.1). Значение, полученное в результате считывания сигнала с потенциометра (в диапазоне от 0 до 1023), масштабируется в диапазон от 0 до 180 градусов.

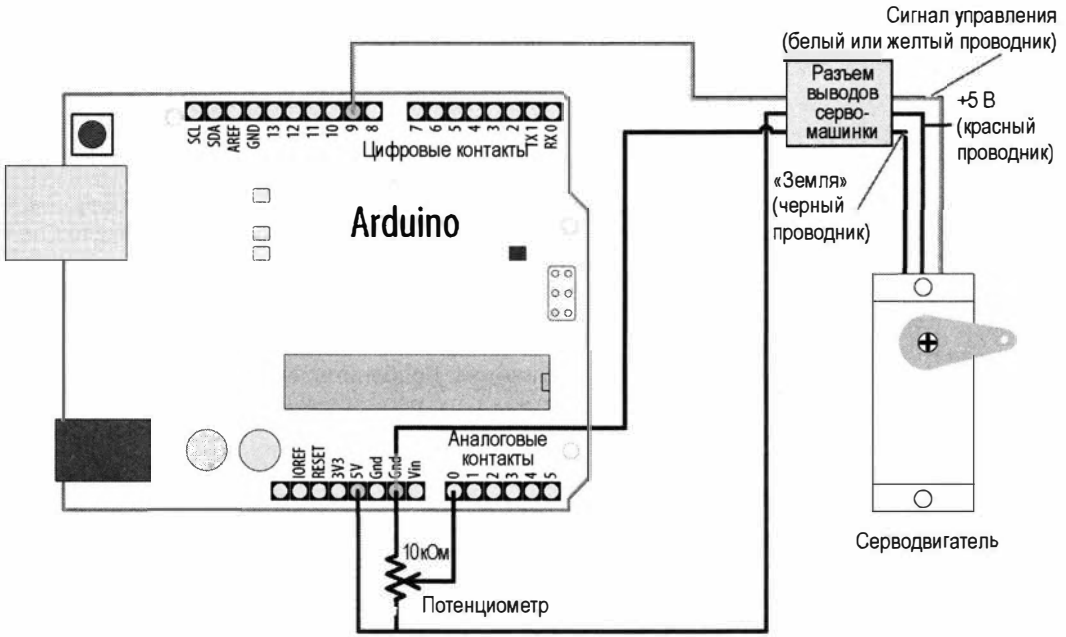


Рис. 8.4. Управление сервомашинкой с помощью потенциометра

Листинг 8.2. Управление сервомашинкой с помощью потенциометра или другого датчика

```
/*
* Скетч Servo With Sensor
* Управляет сервомашинкой с помощью потенциометра или другого датчика
*/

#include <Servo.h>

Servo myservo;   // Создаем экземпляр объекта Servo для управления сервомашинкой

int potpin = A0; // Аналоговый контакт для подачи сигнала потенциометра
int val;         // Переменная для хранения значения, полученного
                 // считыванием сигнала с аналогового контакта

void setup()
{
   myservo.attach(9); // Подключаем к объекту Servo сервомашинку,
                      // подключенную к контакту 9 платы Arduino
}

void loop()
{
   val = analogRead(potpin);          // Считываем значение
                                      // выходного сигнала потенциометра
```

```
    val = map(val, 0, 1023, 0, 180); // Масштабируем его для использования
                                     // с сервомашинкой
    myservo.write(val);              // Подаем масштабированное значение
                                     // на сервомашинку
    delay(15);                       // Пауза, чтобы дать сервомашинке
                                     // время отреагировать на команду
}
```

Выводы любительских сервомашинок оснащаются трехконтактным гнездовым разъемом, который можно напрямую подключать к штыревому разъему, которым оснащаются некоторые шилды управления электродвигателем, — например, Adafruit Motor Shield. Разъем выводов сервомашинки совместим с разъемами контактов платы Arduino, что позволяет использовать с ним самые обычные проволочные перемычки. Следует иметь в виду, что цвет проводника вывода сигнала не стандартизован — иногда вместо белого цвета используется желтый. Красный проводник плюса питания всегда находится посередине, а проводник «земли» обычно черного или коричневого цвета.

Обсуждение работы решения и возможных проблем

В качестве сигнала управления сервомашинки можно использовать любой сигнал, который можно считать с помощью функции `analogRead()` (см. *главу 5* и *6*). В частности, сервомашинкой можно управлять, используя решения по работе с акселерометром из *главы 6*, управляя углом поворота вала сервомашинки поворотом гироскопа или углом наклона акселерометра.

Не все сервомашинки могут перемещать вал по всему диапазону библиотеки Servo. Если в крайней точке диапазона перемещений вал упирается в концевой ограничитель и сервомашинка начинает жужжать, попробуйте уменьшать выходной диапазон функции `map()`, пока жужжание не прекратится. Например:

```
val=map(val,0,1023,10,170);
```

8.3. Управление скоростью вращения сервомашинок непрерывного вращения

ЗАДАЧА

Требуется управлять скоростью и направлением вращения вала сервомашинки, модифицированной для непрерывного вращения. Например, изменять скорость и направление вращения вала сервомашинок, используемых в качестве приводов колес шасси робота.

РЕШЕНИЕ

Сервомашинки непрерывного вращения представляют собой тип электродвигателя с шестеренчатым редуктором, способный вращать свой вал в обоих направлениях с разной скоростью. Сервомашинки непрерывного вращения управляются подобно

обычным сервомашинкам. Когда угол поворота превышает 90 градусов, вал сервомашинки вращается в одном направлении, а когда меньше 90 градусов — в другом. Действительное направление движения платформы с сервомашинками вперед или назад зависит от способа крепления сервомашинок к платформе. На рис. 8.5 показано, как подключить две сервомашинки к плате Arduino.

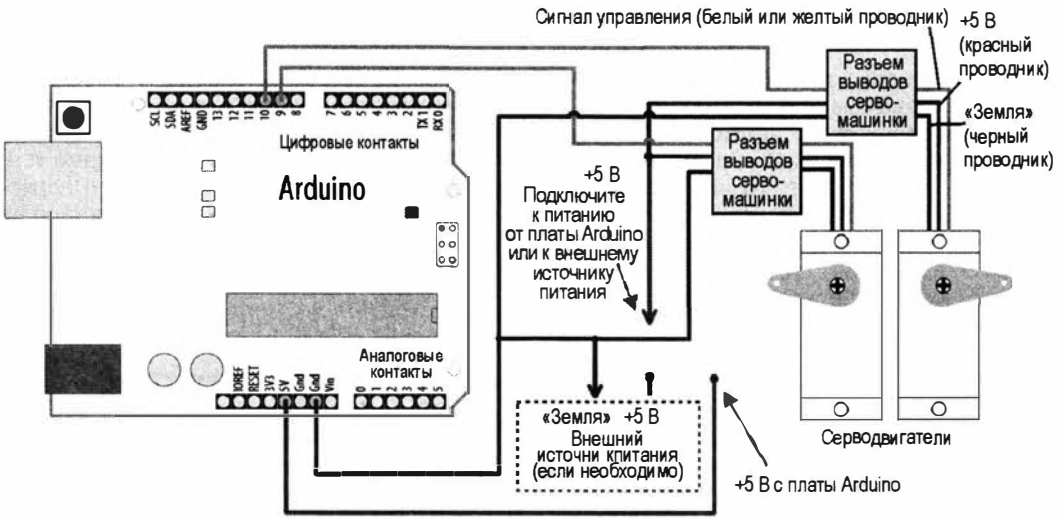


Рис. 8.5. Подключение двух сервомашинок к плате Arduino

Для сервомашинок обычно требуется питание напряжением от 4,8 до 6 вольт. Более мощным сервоприводам может потребоваться ток большей силы, чем плата Arduino может предоставить на контакте выходного напряжения 5V, и для таких сервоприводов нужно использовать отдельный источник питания. В качестве такого отдельного источника питания можно взять четыре аккумулятора напряжением 1,2 В каждый. Но если вы намереваетесь также питать от такой аккумуляторной батареи и плату Arduino, следует иметь в виду, что вы вступаете в «серую зону». В частности, теоретически плату Arduino можно было бы запитать от этой батареи, подключив ее положительный полюс к контакту 5V платы Arduino. Но тогда питание на плату будет подаваться в обход стабилизатора напряжения, что не есть хорошо. Можно также подключить плюс батареи на контакт Vin платы Arduino. Но и эта мысль далека от замечательной, поскольку на этот контакт нужно подавать напряжение величиной минимум 6 В, а напряжение меньше 7 В может вызвать нестабильную работу платы. Поэтому следует учесть эти ограничения и токовые требования используемых сервоприводов, чтобы одно не мешало другому.

В листинге 8.3 приводится скетч для управления двумя сервомашинками непрерывного вращения. Скетч посылает на сервомашинки варьирующийся сигнал управления для поворота от угла 90 градусов до угла 180 градусов, что для сервомашинок непрерывного вращения означает варьирующуюся скорость вращения. Таким образом, если сервомашинки вращают колеса шасси робота, оно начнет двигаться вперед, постепенно повышая скорость, а затем замедляя ее до полной оста-

новки. Поскольку код управления сервомашинками исполняется в цикле, эти действия будут продолжаться до тех пор, пока подается питание.

Листинг 8.3. Управление двумя сервомашинками непрерывного вращения

```
/*
* Скетч Continuous rotation
*/

#include <Servo.h>

Servo myservoLeft;  // Создаем экземпляр объекта Servo для управления
                    // левой сервомашинкой
Servo myservoRight; // Создаем экземпляр объекта Servo для управления
                    // правой сервомашинкой

int angle = 0; // Переменная для хранения позиции сервомашинки

void setup()
{
   myservoLeft.attach(9);   // Подключаем к объекту Servo
        // левую сервомашинку, подключенную к контакту 9 платы Arduino
   myservoRight.attach(10); // Подключаем к объекту Servo
        // правую сервомашинку, подключенную к контакту 10 платы Arduino
}

void loop()
{
   for(angle = 90; angle < 180; angle += 1) // Поворот от 90 градусов
                                            // до 180 градусов
   {
      // шагами величиной в 1 градус
      // Угол в 90 градусов останавливает вращение
      myservoLeft.write(angle);      // Вращаем сервомашинку
                                     // со скоростью, заданной углом
      myservoRight.write(180-angle); // Вращаем в обратном направлении
      delay(20); // Пауза в 20 мс между командами
                 // управления сервомашинкой
   }
   for(angle = 180; angle >= 90; angle -= 1) // Поворот от 180 градусов
                                             // до 90 градусов
   {
      myservoLeft.write(angle);      // Вращаем сервомашинку
                                     // со скоростью, заданной углом
      myservoRight.write(180-angle); // Вращаем в обратном направлении
   }
}
```

Обсуждение работы решения и возможных проблем

Подобный код можно использовать для управления сервомашинками непрерывного вращения и обычными сервомашинками, но при этом следует иметь в виду, что сервомашинки непрерывного вращения могут не остановиться при сигнале управления, соответствующем ровно 90 градусам поворота. Некоторые сервомашинки оснащены подстроечным потенциометром, которым можно откорректировать угол остановки, или же можно просто добавить или отнять несколько градусов, пока не будет получено значение угла, при котором сервомашинка останавливается. Например, если левая сервомашинка останавливается при угле 92 градуса, строку кода управления сервомашинкой можно модифицировать следующим образом:

```
myservoLeft.write(angle+TRIM); // сделайте объявление
                               // int TRIM = 2; в начале скетча
```

8.4. Управление сервомашинками с компьютера

ЗАДАЧА

Требуется управлять сервомашинкой командами, поступающими на плату Arduino через последовательный интерфейс. Например, отдавать команды сервомашинке из программы, исполняющейся на персональном компьютере.

РЕШЕНИЕ

Сервомашинками можно управлять программами, исполняющимися на компьютере. Этот способ обладает тем достоинством, что позволяет управлять любым количеством сервомашинок. Но при этом программа должна постоянно заниматься обновлением позиций сервомашинок, в результате чего ее логика может быстро усложниться при использовании большого количества сервомашинок, если ей при этом нужно выполнять и другие задачи.

В листинге 8.4 приводится пример скетча для управления четырьмя сервомашинками, подключенными к контактам платы Arduino с 7-го по 10-й. Скетч управляет сервомашинками, выполняя команды, получаемые через последовательный порт. Команды имеют следующий формат:

- `180a` — угол 180 для сервомашинки `a`;
- `90b` — угол 90 для сервомашинки `b`;
- `0c` — угол 0 для сервомашинки `c`;
- `17d` — угол 17 для сервомашинки `d`.

Листинг 8.4. Управление сервомашинками командами, получаемыми по последовательному интерфейсу

```
/*
* Скетч Computer Control of Servos
*/
```

```
#include <Servo.h> // Подключаем библиотеку Servo
#define SERVOS 4   // Количество сервомашинок

int servoPins[SERVOS] = {7,8,9,10}; // Сервомашинки подключены
                                    // к контактам 7 по 10

Servo myservo[SERVOS];

void setup()
{
   Serial.begin(9600);
   for(int i=0; i < SERVOS; i++)
   {
      myservo[i].attach(servoPins[i]);
   }
}

void loop()
{
   serviceSerial();
}

// Функция serviceSerial() проверяет наличие данных в последовательном
// порту и обновляет позиции сервомашинок полученными данными
// Данные должны быть следующего вида:
//
// "180a" – угол 180 для сервомашинки a
// "90b – угол 90 для сервомашинки b
//
void serviceSerial()
{
   if ( Serial.available())
   {
      int pos = Serial.parseInt();
      char ch = Serial.read();
      if(ch >= 'a' && ch < 'a' + SERVOS) // Если значение переменной ch
                        // является действительной буквой сервомашинки,
      {
         Serial.print("Servo "); Serial.print(ch - 'a');
         Serial.print(" set to "); Serial.println(pos);
         myservo[ch - 'a'].write(pos); // подаем сигнал позиции
                              // на соответствующую сервомашинку
      }
   }
}
```

Обсуждение работы решения и возможных проблем

Сервомашинки для этого решения подключаются подобно тому, как это делалось в предыдущих решениях. Линия сигнала управления каждой сервомашинки подключается к цифровому контакту платы Arduino. Линии «земли» всех сервомашинок подключаются к контакту «земли» платы Arduino. Линии питания сервомашинок соединяются вместе и, если их токопотребление превышает токовые возможности источника питания платы Arduino, подключаются к внешнему источнику питания напряжением 5–6 вольт.

Массив `myservo` (массивы подробно рассматриваются в *разд. 2.4*) используется для хранения ссылок на четыре сервомашинки. Цикл `for` в функции `setup()` подключает каждую сервомашинку в массиве к последовательным контактам платы Arduino, определенным в массиве `servoPins`.

Для получения целочисленных значений (типа `int`) через последовательный порт используется функция `parseInt()`. При получении буквы *a* (английской) выполняется запись позиции в первую сервомашинку массива (подключенную к контакту 7). Буквы *b*, *c* и *d* управляют записью в последующие сервомашинки.

Дополнительная информация

В *главе 4* приводится подробная информация по обработке значений, получаемых по последовательному интерфейсу.

8.5. Управление бесщеточным электродвигателем с помощью любительского контроллера

ЗАДАЧА

Требуется управлять скоростью вращения бесщеточного электродвигателя такого типа, как используемый в радиоуправляемых моделях самолетов и т. п.

РЕШЕНИЕ

Для этого решения используется скетч из решения, приведенного в *разд. 8.2* (см. листинг 8.2). Используемые устройства подключаются так же, как и в решении из *разд. 8.2*, с той лишь разницей, что вместо сервомашинки к плате Arduino подключается контроллер бесщеточного электродвигателя (рис. 8.6). Любительский электронный контроллер скорости вращения (ESC[1]) представляет собой устройство для управления бесщеточными электродвигателями, используемыми в моделях радиоуправляемых самолетов, квадрокоптеров и т. п. Благодаря массовому производству этих устройств они имеют низкую стоимость, что делает их доступными широкому кругу конструкторов-любителей для управления бесщеточными электро-

[1] От *англ.* Electronic Speed Controller.

двигателями в своих моделях. Подходящий вам контроллер ESC можно найти, выполнив поиск по ключевому слову `esc` или фразе `electronic speed controller` на вебсайте магазина оборудования для моделирования или таких веб-магазинов, как Amazon, eBay, AliExpress и т. п.

Бесщеточные электродвигатели имеют три вывода, которые подключаются к контроллеру ESC согласно документации (справочный листок, datasheet) для него. На рис. 8.6 показан пример подключения бесщеточного электродвигателя к контроллеру ESC. Неплохо также посмотреть в документации на контроллер ESC, нет ли для него каких-либо особых требований при работе с платформой Arduino. Например, для контроллера ESC компании RC Electric Parts (**https://oreil.ly/c90yi**) рекомендуется инициализировать объект Servo следующим образом:

```
Servo.attach(pin, 1000, 2000);
```

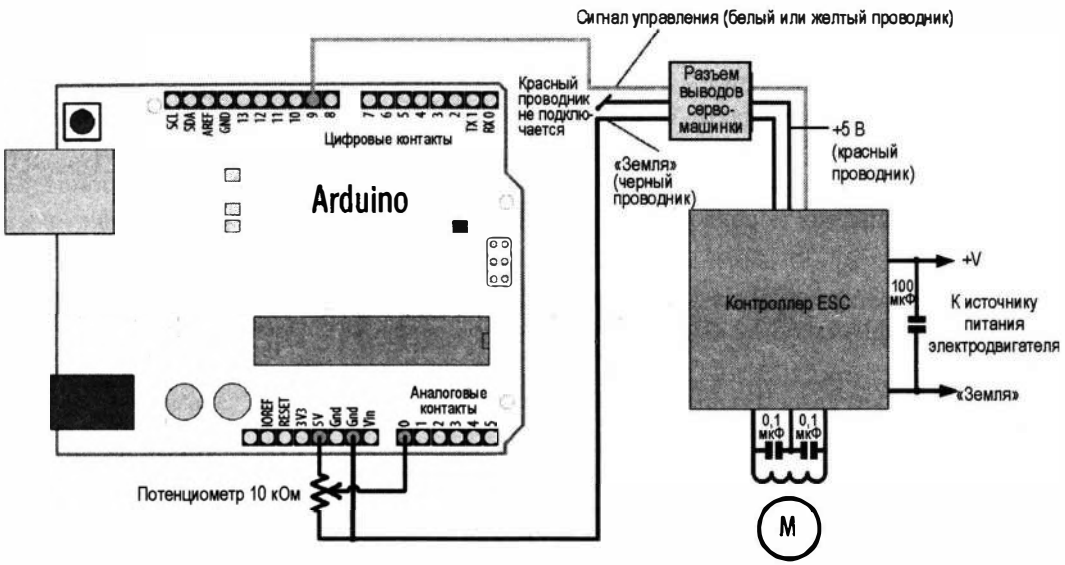


Рис. 8.6. Подключение контроллера ESC к плате Arduino

Обсуждение работы решения и возможных проблем

Прежде чем подключать контроллер ESC к плате Arduino, посмотрите в его справочном листке, подходит ли он для используемого бесщеточного электродвигателя, а также уточните, как именно подключать к нему электродвигатель. Контроллеры ESC обычно имеют три вывода с одного конца — для подключения к электродвигателю и две группы выводов с другого конца. Одна группа из двух выводов служит для подключения к источнику питания, а другая — из трех выводов и оснащенная разъемом — для подключения к устройству управления, например контроллеру полета или плате Arduino. На средний из трех выводов для подключения к устройству управления обычно выводится +5 В напряжения питания. Этот вывод нужно отключить или обрезать, если только он не будет использоваться для подачи питания на плату Arduino.

Если используемый контроллер ESC оснащен компонентом, предоставляющим +5 В напряжения для питания сервомашинок или других устройств (такой компонент называется схемой BEC[2]), этот вывод не подключается к плате Arduino (см. рис. 8.6).

8.6. Управление соленоидами и реле

ЗАДАЧА

Требуется программным способом управлять соленоидом или реле. *Соленоиды* — это электромагнитные устройства, которые преобразовывают электрическую энергию в механическое поступательное движение. А *электромагнитное реле* представляет собой переключатель, управляемый соленоидом.

РЕШЕНИЕ

Для большинства соленоидов требуется ток большей силы, чем может быть предоставлен контактами платы Arduino, поэтому для управления этими устройствами применяются транзисторы. Схема подключения соленоида к плате Arduino с использованием транзистора показана на рис. 8.7. Для активирования соленоида на контакте управления посредством функции `digitalWrite()` устанавливается высокий уровень (`HIGH`).

В листинге 8.5 приводится скетч для управления соленоидом, который включает соленоид на одну секунду каждый час.

Листинг 8.5. Скетч для управления соленоидом с использованием транзистора

```
/*
* Скетч Solenoid
*/

int solenoidPin = 2; // Соленоид, подключенный через транзистор к контакту 2

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

void loop()
{
   long interval = 1000 * 60 * 60 ;  // interval = 60 минут
```

[2] От *англ.* Battery Eliminator Circuit, схема устранения батареи. Для питания бесщеточных электродвигателей обычно используется напряжение 12 В или более, что не подходит для питания остальных компонентов модели, для которых обычно требуется напряжение питания 5 В. Вместо использования отдельной батареи на 5 В это питание берется со схемы BEC контроллера ESC.

```
    digitalWrite(solenoidPin, HIGH);  // Включаем соленоид
    delay(1000);                      // Пауза длительностью в 1 секунду
    digitalWrite(solenoidPin, LOW);   // Выключаем соленоид
    delay(interval);                  // Пауза длительностью в 1 час
}
```

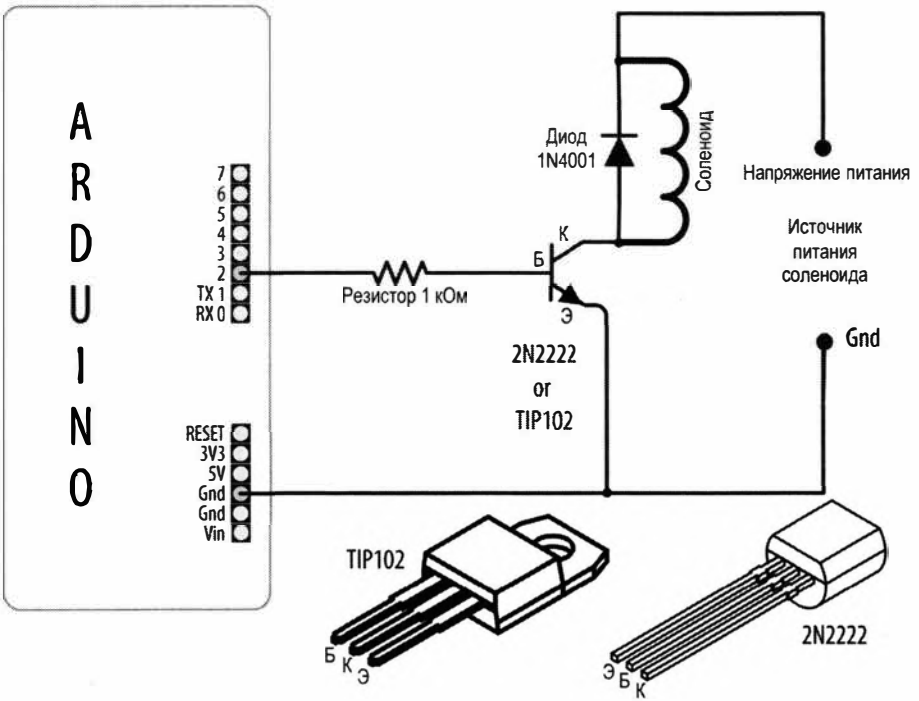


Рис. 8.7. Подключение соленоида к плате Arduino с использованием транзистора

Обсуждение работы решения и возможных проблем

Выбор транзистора для управления соленоидом зависит от силы тока, требуемого для его включения (активирования). В справочном листке эта характеристика может указываться в миллиамперах или как сопротивление обмотки. В последнем случае требуемый ток определяют, разделив рабочее напряжение обмотки в вольтах на ее сопротивление в омах. Например, величина потребляемого тока реле с рабочим напряжением 12 В и сопротивлением 185 Ом будет равна 65 мА:

```
12 В / 185 Ом = 0,065 А или 65 мА
```

Если используемый транзистор не рассчитан на протекающий через него ток, он будет греться и может в конечном счете сгореть.

Для соленоидов, потребляющих ток величиной до нескольких сотен миллиампер, можно использовать маломощный транзистор — например, 2N222. Для управления более мощными соленоидами потребуется более мощный транзистор — например, TIP102/TIP120 или подобный. Кроме указанных транзисторов можно использовать

и другие эквивалентные транзисторы. В *приложении 2* показано, как пользоваться справочным листком при выборе транзистора с требуемыми характеристиками.

Диод, подключенный к выводам обмотки соленоида, предохраняет транзистор от повреждения создаваемой обмоткой обратной ЭДС (обратная электродвижущая сила — напряжение, создаваемое в обмотке при отключении протекающего через нее тока). Важно соблюдать правильную полярность при подключении защитного диода: вывод катода (обозначенный полоской белого или иного цвета) подключается к плюсу источника питания соленоида.

Электромагнитные реле управляются точно таким же образом, что и соленоиды. При этом полупроводниковые реле собраны на полупроводниковых электронных схемах, что позволяет подключать их непосредственно к контактам платы Arduino без необходимости использовать промежуточный транзистор. Прежде чем подключать реле к плате Arduino, проверьте в справочном листке на него его рабочее напряжение и ток. Если при напряжении 5 В рабочий ток реле превышает 40 мА, такое реле необходимо подключать к плате Arduino через транзистор, как показано на рис. 8.7.

8.7. Вибрация объекта

ЗАДАЧА

Требуется организовать вибрацию объекта под управлением скетча Arduino. Например, каждую минуту включать вибрацию объекта в течение одной секунды.

РЕШЕНИЕ

Эта задача решается с помощью вибрационного электродвигателя, подключенного к плате Arduino, как показано на рис. 8.8. Конденсатор емкостью 0,1 мкФ может

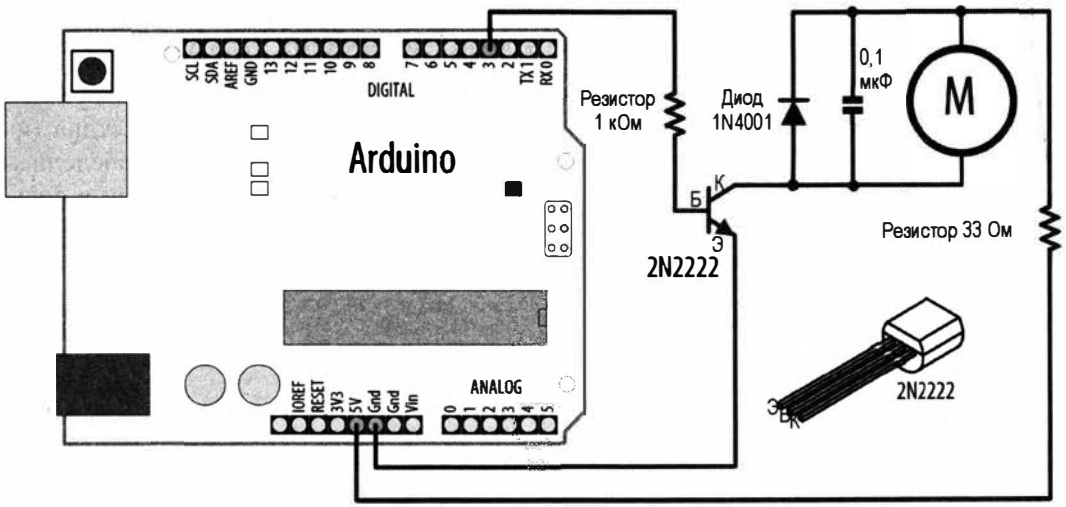


Рис. 8.8. Подключение вибрационного электродвигателя к плате Arduino

быть керамическим или электролитным, но в последнем случае его положительный вывод нужно в обязательном порядке подключить к резистору номиналом 33 Ом. Другой вывод конденсатора подключен к контакту 5V платы Arduino.

Скетч в листинге 8.6 включает вибрационный электродвигатель на одну секунду каждую минуту.

Листинг 8.6. Управление вибрационным электродвигателем

```
/*
* Скетч Vibrate
* Включает вибрационный электродвигатель на одну секунду каждую минуту
*/

const int motorPin = 3; // Вибрационный электродвигатель,
                        // подключен через транзистор к контакту 3

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

void loop()
{
   digitalWrite(motorPin, HIGH); // Включаем вибродвигатель
   delay(1000);                  // Задержка длительностью в 1 минуту
   digitalWrite(motorPin, LOW);  // Выключаем вибродвигатель
   delay(59000);                 // Пауза длительностью в 59 секунд
}
```

Обсуждение работы решения и возможных проблем

В решении используется специальный тип электродвигателя — *вибродвигатель* (например, вибродвигатель артикул ROB-08449 компании SparkFun). Вибродвигатель можно также извлечь из старого сотового телефона, оснащенного функцией вибровызова. Подобно соленоидам, для питания вибродвигателей требуется ток большей силы, чем может быть предоставлен контактами платы Arduino, поэтому для управления этими устройствами используются транзисторы. Вы можете применить практически любой NPN-транзистор — например, распространенный транзистор 2N2222, как показано в схеме подключения вибродвигателя на рис. 8.8. Сигнал управления с контакта Arduino подается на базу транзистора через резистор номиналом 1 кОм. Резистор служит для ограничения тока, протекающего через контакт управления. Точное значение номинала этого резистора не критично, и можно использовать значения вплоть до 4,7 кОм или около того. Диод, подключенный параллельно обмотке вибродвигателя, служит для подавления напряжений, создаваемых обмоткой при вращении вибродвигателя (такой диод иногда называют *демпфирующим*). А подключенный параллельно диоду конденсатор подавляет

всплески напряжения, создаваемые замыканием и размыканием щеток вибродвигателя (контактов, подающих питание на коллектор его ротора). Резистор номиналом 33 Ом служит для ограничения протекающего через вибродвигатель тока.

Скетч решения устанавливает на контакте управления высокий уровень (HIGH) на одну секунду, после чего выдерживает паузу в 59 секунд, бесконечно повторяя этот процесс. С контакта управления высокий уровень подается на базу транзистора, включая его и разрешая протекание тока через обмотку электродвигателя.

В листинге 8.7 приводится альтернативная версия скетча, управляющая включением вибродвигателя на основе значения сигнала, считанного с фоторезисторного датчика. Вибродвигатель подключается так же, как и в схеме на рис. 8.8, а фоторезистор подключается к аналоговому контакту A0 (см. рис. 6.2).

Листинг 8.7. Управление вибродвигателем посредством фоторезистора

```
/*
* Скетч Vibrate_Photocell
* Включает вибродвигатель при определенном уровне освещения
*/

const int motorPin = 3;    // Вибрационный электродвигатель
                           // подключен через транзистор к контакту 3
const int sensorPin = A0; // Фоторезистор подключен
                           // к аналоговому контакту A0
int sensorAmbient = 0;     // Уровень освещения окружающего
                           // пространства (калибрируется в функции setup())
const int thresholdMargin = 100; // Уровень освещения для включения вибродвигателя

void setup()
{
   pinMode(motorPin, OUTPUT);
   sensorAmbient = analogRead(sensorPin); // Получаем начальный уровень освещения
}

void loop()
{
   int sensorValue = analogRead(sensorPin);
   if( sensorValue > sensorAmbient + thresholdMargin)
   {
      digitalWrite(motorPin, HIGH); // Включаем вибродвигатель
   }
   else
   {
      digitalWrite(motorPin, LOW); // Выключаем вибродвигатель
   }
}
```

Этот скетч при освещении фоторезистора устанавливает на контакте управления вибродвигателем высокий уровень. В начале скетча считывается уровень фонового освещения и полученное значение сохраняется в качестве порогового в переменной `sensorAmbient`. При исполнении кода в главном цикле `loop()` каждое событие превышения фонового уровня освещения будет включать вибродвигатель.

8.8. Управление щеточным электродвигателем с использованием транзистора

ЗАДАЧА

Требуется включать и выключать щеточный электродвигатель, а также управлять скоростью его вращения. Двигатель вращается только в одном направлении.

РЕШЕНИЕ

Скетч в листинге 8.8 включает и выключает электродвигатель и управляет скоростью его вращения, реагируя на команды, получаемые по последовательному интерфейсу. Схема для подключения щеточного электродвигателя к плате Arduino показана на рис. 8.9. Конденсатор емкостью 0,1 мкФ может быть керамическим или электролитным, но в последнем случае его положительный вывод нужно в обязательном порядке подключить к плюсу источника питания электродвигателя.

Листинг 8.8. Управление щеточным электродвигателем

```
/*
* Скетч SimpleBrushed
* Выполняет для управления электродвигателем команды, получаемые
  по последовательному интерфейсу,
* Действительные команды: цифры от 0 по 9, где 0 означает
  выключенное состояние, а 9 - максимальную скорость
*/

const int motorPin = 3; // Электродвигатель подключается
                        // через транзистор к контакту 3

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

void loop()
{
   if (Serial.available())
   {
      char ch = Serial.read();
```

```
      if(isDigit(ch)) // Значение переменной ch цифра?
      {
         int speed = map(ch, '0', '9', 0, 255);
         analogWrite(motorPin, speed);
         Serial.println(speed);
      }
      else
      {
         Serial.print("Unexpected character "); // Неизвестный символ
         Serial.println(ch);
      }
   }
}
```

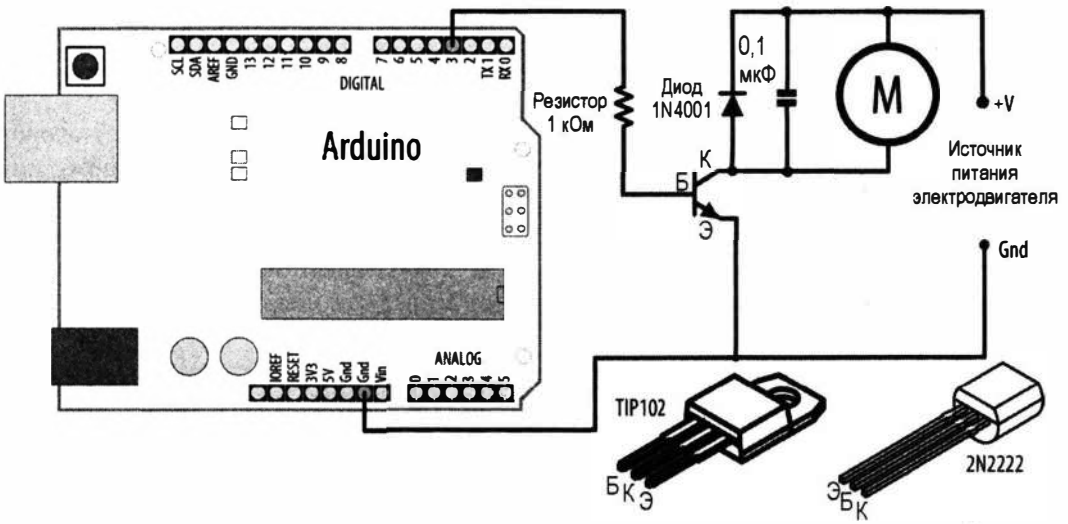


Рис. 8.9. Подключение щеточного электродвигателя к плате Arduino

Обсуждение работы решения и возможных проблем

Это решение похоже на решение из *разд. 8.7* с той лишь разницей, что для управления скоростью вращения электродвигателя используется функция `analogWrire()`. Функция `analogWrite()` и широтно-импульсная модуляция подробно рассматриваются в *разд. «Аналоговый вывод» главы 7*.

8.9. Управление направлением вращения щеточного электродвигателя с помощью Н-моста

ЗАДАЧА

Требуется управлять направлением вращения щеточного электродвигателя — например, подавая соответствующие команды по последовательному интерфейсу.

РЕШЕНИЕ

Эта задача решается с помощью устройства под названием *Н-мост*, которое позволяет менять на обратное полярность подаваемого на электродвигатель питания, изменяя таким образом направление его вращения. Кроме того, оно позволяет полностью останавливать вращение. Название «Н-мост» происходит от внешнего вида принципиальной схемы устройства на дискретных компонентах (механических или транзисторных переключателях), которая похожа на букву «Н». Однако в рассматриваемом решении используется микросхема L293D Н-моста, которая может управлять двумя щеточными электродвигателями. На рис. 8.10 показана принципиальная схема подключения микросхемы к плате Arduino и электродвигателю. Вместо этой микросхемы можно также использовать микросхему SN754410, которая имеет точно такую же цоколевку. Конденсаторы номиналом 0,1 мкФ на электродвигателях должны быть керамическими. Скетч для управления одним двигателем этой схемы приводится в листинге 8.9.

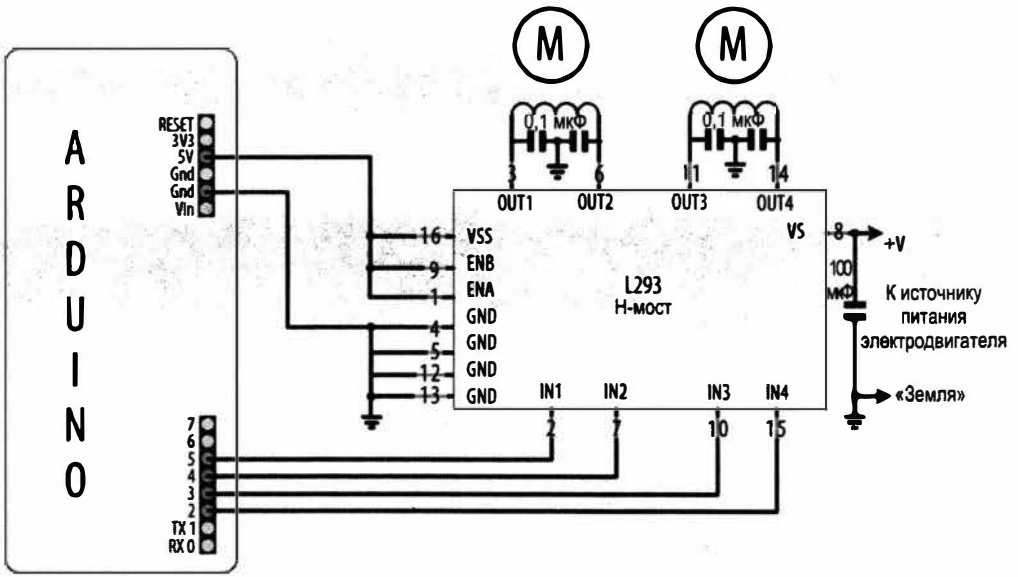


Рис. 8.10. Подключение двух щеточных электродвигателей к плате Arduino через Н-мост L293D

Листинг 8.9. Управление щеточным электродвигателем с помощью Н-моста

```
/*
* Скетч Brushed_H_Bridge_simple
* Выполняет команды, получаемые по последовательному интерфейсу
  для изменения направления вращения щеточного электродвигателя
* Направление вращения задается символами + и -. Любой другой символ
  останавливает электродвигатель
*/
```

```
const int in1Pin = 5; // Контакты для управления Н-мостом
const int in2Pin = 4;

void setup()
{
   Serial.begin(9600);
   pinMode(in1Pin, OUTPUT);
   pinMode(in2Pin, OUTPUT);
   Serial.println("+ - to set direction, any other key stops motor");
                  // Нажмите + или -, чтобы задать направление вращения
                  // Любой другой символ останавливает электродвигатель
}

void loop()
{
   if ( Serial.available())
   {
      char ch = Serial.read();
      if (ch == '+')
      {
         Serial.println("CW"); // По часовой стрелке
         digitalWrite(in1Pin,LOW);
         digitalWrite(in2Pin,HIGH);
      }
      else if (ch == '-')
      {
         Serial.println("CCW");  // Против часовой стрелки
         digitalWrite(in1Pin,HIGH);
         digitalWrite(in2Pin,LOW);
      }
      else if (ch != '\n' && ch != '\r') // Игнорируем символы возврата
                                        // каретки и новой строки
      {
         Serial.print("Stop motor");  // Остановить электродвигатель
         digitalWrite(in1Pin,LOW);
         digitalWrite(in2Pin,LOW);
      }
   }
}
```

Обсуждение работы решения и возможных проблем

В табл. 8.1 приводятся комбинации значений сигналов управления, подаваемые с платы Arduino на Н-мост, и эффект их воздействия на электродвигатель. Скетч этого решения управляет одним электродвигателем, используя контакты IN1 и IN2 микросхемы Н-моста. На контакт EN подается постоянный высокий уровень, поскольку он подключен к напряжению +5 В.

Таблица 8.1. Комбинации сигналов управления H-мостом

EN	IN1	IN2	Функция
HIGH	LOW	HIGH	Вращение по часовой стрелке
HIGH	HIGH	LOW	Вращение против часовой стрелки
HIGH	LOW	LOW	Остановка электродвигателя
HIGH	HIGH	HIGH	Остановка электродвигателя
LOW	Игнорируется	Игнорируется	Остановка электродвигателя

В схеме на рис. 8.10 показано подключение двух электродвигателей. Скетч для управления ими приводится в листинге 8.10.

Листинг 8.10. Управление двумя электродвигателями с помощью H-моста

```
/*
* Скетч Brushed_H_Bridge_simple2
* Выполняет команды, получаемые по последовательному интерфейсу
  для изменения направления вращения щеточных электродвигателей
* Направление вращения задается символами + и -. Любой другой символ
  останавливает электродвигатели
*/

const int in1Pin = 5; // Контакты для управления первым H-мостом
const int in2Pin = 4;
const int in3Pin = 3; // Контакты для управления вторым H-мостом
const int in4Pin = 2;

void setup()
{
   Serial.begin(9600);
   pinMode(in1Pin, OUTPUT);
   pinMode(in2Pin, OUTPUT);
   pinMode(in3Pin, OUTPUT);
   pinMode(in4Pin, OUTPUT);
   Serial.println("+ - sets direction of motors, any other key stops motors");
      // Направление вращения задается символами + и -
      // Любой другой символ останавливает электродвигатели
}

void loop()
{
   if ( Serial.available())
   {
      char ch = Serial.read();
      if (ch == '+')
```

```
    {
      Serial.println("CW"); // По часовой стрелке
      // Первый электродвигатель
      digitalWrite(in1Pin,LOW);
      digitalWrite(in2Pin,HIGH);
      //Второй электродвигатель
      digitalWrite(in3Pin,LOW);
      digitalWrite(in4Pin,HIGH);
    }
    else if (ch == '-')
    {
      Serial.println("CCW"); // Против часовой стрелки
      digitalWrite(in1Pin,HIGH);
      digitalWrite(in2Pin,LOW);
      digitalWrite(in3Pin,HIGH);
      digitalWrite(in4Pin,LOW);
    }
    else if (ch != '\n' && ch != '\r') // Игнорируем символы
                                       // возврата каретки и новой строки
    {
      Serial.print("Stop motors"); // Остановить электродвигатели
      digitalWrite(in1Pin,LOW);
      digitalWrite(in2Pin,LOW);
      digitalWrite(in3Pin,LOW);
      digitalWrite(in4Pin,LOW);
    }
  }
}
```

8.10. Управление направлением и скоростью вращения щеточного электродвигателя с помощью Н-моста

ЗАДАЧА

Организовать управление направлением вращения и скоростью вращения щеточного электродвигателя с помощью Н-моста.

РЕШЕНИЕ

Это решение расширяет функциональность решения из *разд. 8.9*, добавляя к возможности управления направлением вращения щеточного электродвигателя возможность управления также и скоростью его вращения. Подключение микросхемы Н-моста и щеточного электродвигателя к плате Arduino показано на рис. 8.11. Конденсаторы номиналом 0,1 мкФ на электродвигателях должны быть керамическими.

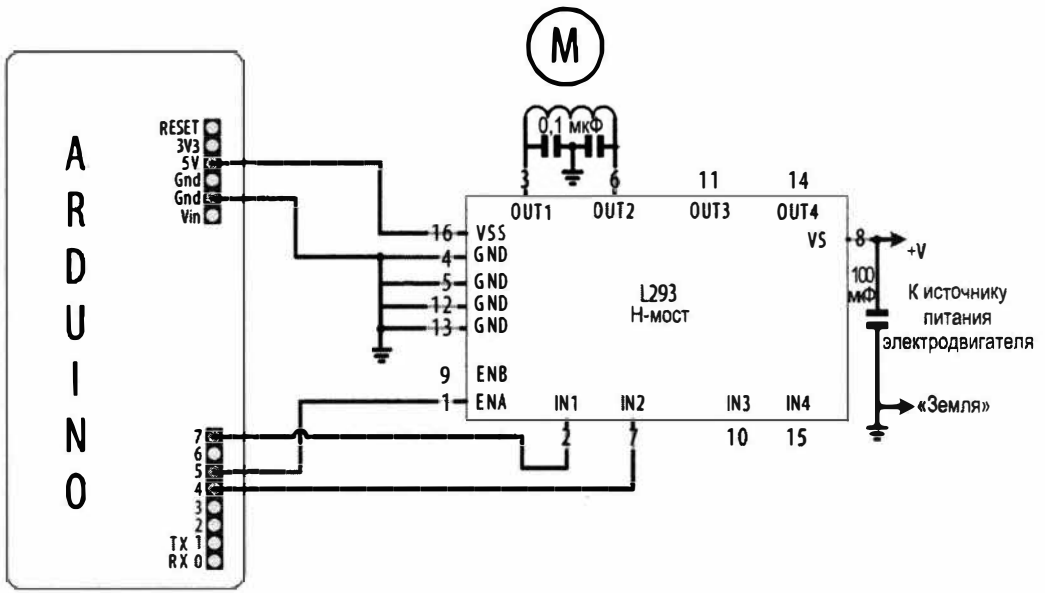


Рис. 8.11. Подключение щеточного электродвигателя к плате Arduino с помощью микросхемы Н-моста

Скетч для управления этой схемой приводится в листинге 8.11. Скетч получает команды, отправляемые с монитора порта, и выполняет их для управления скоростью и направлением вращения электродвигателя. Команда в виде символа `0` (ноль) останавливает электродвигатель, а команды от `1` до `9` задают скорость вращения. Направление вращения задается командами `+` (плюс) и `-` (минус).

Листинг 8.11. Управление скоростью и направлением вращения электродвигателя

```
/*
* Скетч Brushed_H_Bridge with speed control
* Выполняет команды управления скоростью и направлением вращения
  электродвигателя, получаемые по последовательному интерфейсу
* Действительные команды: цифры от 0 по 9, где 0 означает выключенное
  состояние, а 9 - максимальную скорость
* Символы + и - задают направление
*/

const int enPin = 5;  // Контакты для разрешения Н-моста
const int in1Pin = 7; // Контакты для управления Н-мостом
const int in2Pin = 4;

void setup()
{
   Serial.begin(9600);
   pinMode(in1Pin, OUTPUT);
   pinMode(in2Pin, OUTPUT);
```

```
   Serial.println("Speed (0-9) or + - to set direction");
            // Введите с 0 до 9 для скорости, + или - для направления
}

void loop()
{
   if ( Serial.available())
   {
      char ch = Serial.read();
      if(isDigit(ch)) // Значение переменной ch цифра?
      {
         int speed = map(ch, '0', '9', 0, 255);
         analogWrite(enPin, speed);
         Serial.println(speed);
      }
      else if (ch == '+')
      {
         Serial.println("CW");  // По часовой стрелке
         digitalWrite(in1Pin,LOW);
         digitalWrite(in2Pin,HIGH);
      }
      else if (ch == '-')
      {
         Serial.println("CCW");  // Против часовой стрелки
         digitalWrite(in1Pin,HIGH);
         digitalWrite(in2Pin,LOW);
      }
      else if (ch != '\n' && ch != '\r') // Игнорируем символы возврата
                                         // каретки и новой строки
      {
         Serial.print("Unexpected character "); // Неизвестный символ
         Serial.println(ch);
      }
   }
}
```

Обсуждение работы решения и возможных проблем

Это решение похоже на решение из *разд. 8.9* (см. листинг 8.9), в котором направление вращения электродвигателя управляется логическими уровнями на контактах IN1 и IN2 микросхемы H-моста. Кроме этого, приведенное здесь решение управляет также и скоростью вращения электродвигателя, подавая посредством функции `analogWrite()` ШИМ-сигнал на контакт EN для разрешения H-моста (широтно-импульсная модуляция подробно рассматривается в *главе 7*). Запись на этот контакт значения 0 останавливает электродвигатель, а значения 255 — заставляет его вращаться с максимальной скоростью. Значения между этими двумя пределами задают скорость вращения, прямо пропорциональную текущему значению.

8.11. Управление направлением и скоростью вращения щеточного электродвигателя с помощью датчиков

ЗАДАЧА

Требуется управлять направлением и скоростью вращения щеточного электродвигателя, используя сигнал обратной связи с датчиков. Например, организовать управление скоростью и направлением электродвигателей с помощью двух фотодатчиков таким образом, чтобы шасси робота двигалось по направлению к источнику света.

РЕШЕНИЕ

Двигатели для этого решения подключаются так же, как показано на рис. 8.10, но к схеме добавлены два фоторезистора (или фототранзистора — подробную информацию по фототранзисторам см. в *разд. 1.6*), как показано в схеме на рис. 8.12. Конденсаторы номиналом 0,1 мкФ на электродвигателях должны быть керамическими.

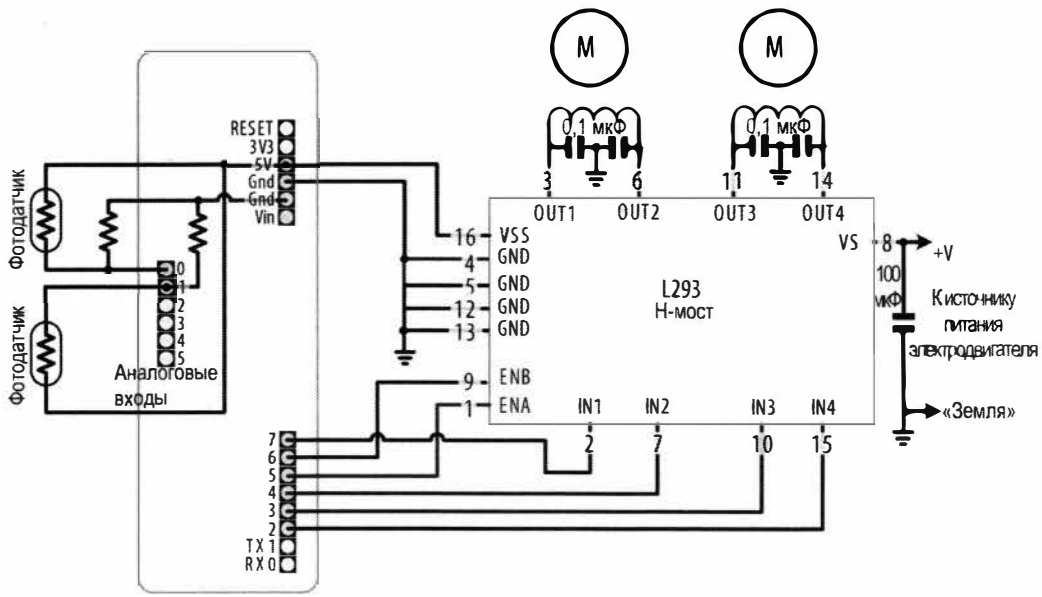


Рис. 8.12. Управление двумя щеточными электродвигателями с помощью датчиков

Скетч для работы с этой схемой приводится в листинге 8.12. Скетч отслеживает уровень освещенности на каждом из фотодатчиков и управляет электродвигателями таким образом, чтобы направлять шасси к датчику с более высоким уровнем освещенности.

Листинг 8.12. Управление двумя электродвигателями с помощью фотодатчиков

```
/*
* Скетч Brushed_H_Bridge_Direction
* Управляет электродвигателями на основании сигналов фотодатчиков,
* направляя шасси робота к источнику света
*/

int leftPins[] = {5,7,4}; // Один контакт для ШИМ (скорость),
            // два контакта для направления левого двигателя
int rightPins[] = {6,3,2}; // То же, для правого двигателя

const int MIN_PWM = 64;  // Это значение может быть в диапазоне от 0 до MAX_PWM
const int MAX_PWM = 128;      // Это значение может быть в диапазоне от 50 до 255
const int leftSensorPin = A0; // Контакты для подключения фотодатчиков
const int rightSensorPin = A1;

int sensorThreshold = 0; // Пороговый уровень освещенности на датчике
                         // для вращения электродвигателя

void setup()
{
   for(int i=1; i < 3; i++)
   {
      pinMode(leftPins[i], OUTPUT);
      pinMode(rightPins[i], OUTPUT);
   }
}

void loop()
{
   int leftVal = analogRead(leftSensorPin);
   int rightVal = analogRead(rightSensorPin);

   if(sensorThreshold == 0) // Датчики откалиброваны?
   {
      // Если нет, калибруем до уровня освещенности несколько выше
      // среднего уровня освещенности окружающей среды
      sensorThreshold = ((leftVal + rightVal) / 2) + 100 ;
   }

   if( leftVal > sensorThreshold || rightVal > sensorThreshold)
   {
      // Если уровень освещенности достаточно высокий
      // для движения вперед
      setSpeed(rightPins, map(rightVal,0,1023, MIN_PWM, MAX_PWM));
      setSpeed(leftPins, map(leftVal ,0,1023, MIN_PWM, MAX_PWM));
   }
}
```

```
void setSpeed(int pins[], int speed )
{
   if(speed < 0)
   {
      digitalWrite(pins[1],HIGH);
      digitalWrite(pins[2],LOW);
      speed = -speed;
   }
   else
   {
      digitalWrite(pins[1],LOW);
      digitalWrite(pins[2],HIGH);
   }
   analogWrite(pins[0], speed);
}
```

Обсуждение работы решения и возможных проблем

Скетч управляет скоростью вращения двух электродвигателей, реагируя на уровень освещенности, определяемый двумя фотодатчиками. Фотодатчики установлены таким образом, что повышение уровня освещенности на одной стороне шасси повышает скорость вращения электродвигателя на другой стороне шасси. В результате шасси будет поворачиваться по направлению к более высокому уровню освещенности. При одинаковом уровне освещенности с обеих сторон шасси будет двигаться вперед по прямой линии. При слишком низком уровне освещенности шасси останавливается.

Если ваше шасси вместо того, чтобы поворачиваться к более яркому источнику света, отворачивается от него, попробуйте заменить полярность подключения питания на обоих электродвигателях. А если шасси крутится на месте, когда оно должно двигаться вперед, переключите полярность питания только одного из электродвигателей.

Сигнал с фотодатчиков подается на аналоговые контакты A0 и A1 платы Arduino, с которых он считывается функцией `analogRead()` (эта функция подробно рассматривается в *разд. 6.3*). В начале исполнения скетча измеряется уровень фоновой освещенности, и это пороговое значение используется для определения минимального уровня освещенности, необходимого, чтобы начинать движение шасси. К измеренному пороговому уровню освещенности добавляется запас величиной в 100 единиц, чтобы небольшие изменения уровня фоновой освещенности не вызывали движения. Значение уровня освещенности, возвращенное функцией `analogRead()`, преобразовывается с помощью функции `map()` в значение ШИМ. Константе `MIN_PWM` следует задать приблизительное значение ШИМ, при котором шасси должно начинать движение (слишком низкие значения не дадут достаточного крутящего момента. Точное значение надо будет определить экспериментальным путем). А константе `MAX_PWM` нужно задать значение (до 255), определяющее максимальную скорость шасси.

Скорость вращения электродвигателей управляется функцией `setSpeed()`. Для управления направлением вращения каждого электродвигателя используются два контакта платы Arduino и еще один контакт — для управления скоростью вращения. Номера этих контактов хранятся в массивах `leftPins` и `rightPins` для левого и правого двигателей соответственно. В первом элементе массива хранится номер контакта для управления скоростью вращения, а в двух других — для управления направлением.

В любом из решений, в котором используется микросхема Н-моста L293, вместо нее можно альтернативно применить микросхему TB6612FNG. На рис. 8.13 показано подключение к плате Arduino микросхемы Н-моста TB6612 (установленной на адаптерной плате артикул 713 компании Pololu).

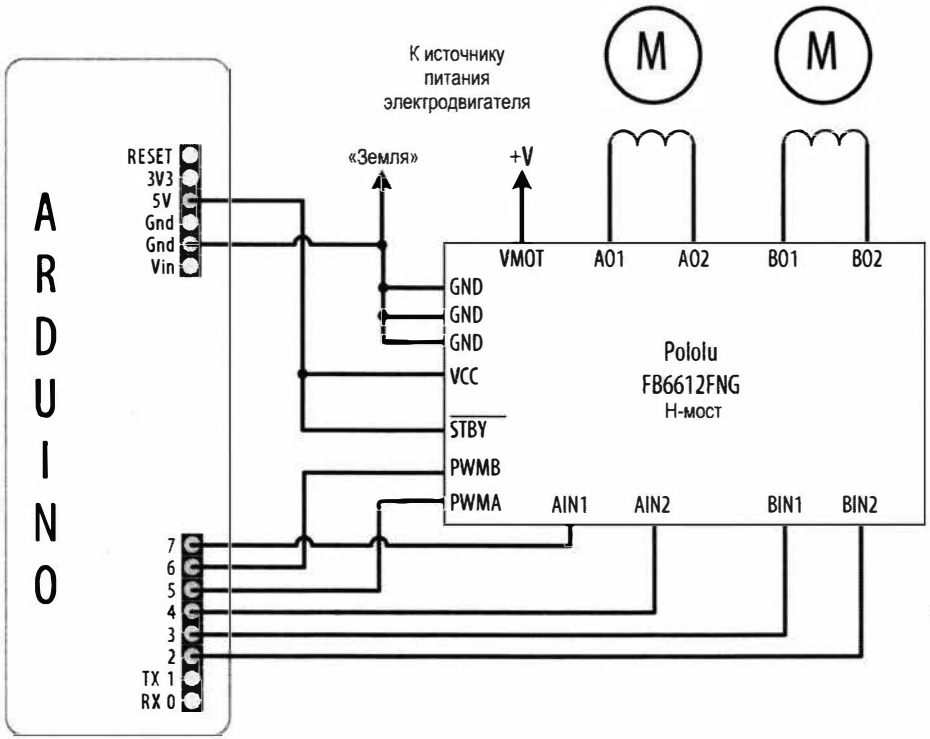


Рис. 8.13. Подключение адаптерной платы микросхемы Н-моста TB6612 производства компании Pololu

Количество контактов платы Arduino, задействованных для управления схемой, можно уменьшить, добавив в схему дополнительный компонент для управления контактами направления Н-моста. Такой компонент представляет собой транзистор или логический элемент для инвертирования логического уровня, подаваемого на другой ввод Н-моста, позволяя использовать только один контакт платы Arduino для управления направлением вращения электродвигателя. Соответствующую принципиальную схему можно найти на веб-сайте Arduino, но также существуют и готовые решения — например, шилд Н-моста Arduino Motor Shield (артикул 7630049200371, **https://store.arduino.cc/arduino-motor-shield-rev3**) или шилд

Ardumoto компании SparkFun (артикул DEV-09213). Оба эти шилда используют микросхему L298, которая является более мощной альтернативой микросхеме L293. Эти шилды вставляются в гнездовые разъемы платы Arduino и требует только подключения к электродвигателю и его источнику питания.

В листинге 8.13 приводится скетч для управления электродвигателями с помощью шилда Arduino Motor Shield (контакты A0 и A1 платы Arduino задействованы для измерения тока, поэтому для управления в скетче используются контакты A2 и A3).

Листинг 8.13. Управление электродвигателями с помощью шилда Arduino Motor Shield

```
/*
* Скетч Brushed_H_Bridge_Direction sketch for motor shield
* Управляет электродвигателями на основании сигналов фотодатчиков,
* направляя шасси робота к источнику света
*/

int leftPins[] = {3,12}; // Один контакт для ШИМ-сигнала (скорость),
                         // один – для направления
int rightPins[] = {11,13};

const int MIN_PWM = 64;       // Это значение может быть в диапазоне
                              // от 0 до MAX_PWM
const int MAX_PWM = 128;      // Это значение может быть в диапазоне от 50 до 255
const int leftSensorPin = A2; // Аналоговые контакты для подключения фотодатчиков
const int rightSensorPin = A3;

int sensorThreshold = 0; // Пороговый уровень освещенности на датчике
                         // для вращения электродвигателя

void setup()
{
   pinMode(leftPins[1], OUTPUT);
   pinMode(rightPins[1], OUTPUT);
}

void loop()
{
   int leftVal = analogRead(leftSensorPin);
   int rightVal = analogRead(rightSensorPin);

   if(sensorThreshold == 0) // Датчики откалиброваны?
   {
      // Если нет, калибруем до уровня освещенности несколько выше
      // среднего уровня освещенности окружающей среды
      sensorThreshold = ((leftVal + rightVal) / 2) + 100 ;
   }
```

```
    if( leftVal > sensorThreshold || rightVal > sensorThreshold)
    {
      // Если уровень освещенности достаточно высокий для движения вперед
      setSpeed(rightPins, map(rightVal,0,1023, MIN_PWM, MAX_PWM));
      setSpeed(leftPins, map(leftVal, 0,1023, MIN_PWM, MAX_PWM));
    }
}

void setSpeed(int pins[], int speed )
{
    if(speed < 0)
    {
      digitalWrite(pins[1], HIGH);
      speed = -speed;
    }
    else
    {
      digitalWrite(pins[1], LOW);
    }
    analogWrite(pins[0], speed);
}
```

Функция `loop()` здесь идентична этой функции скетча из листинге 8.12. Функция `setSpeed()` скетча содержит меньший объем кода, чем ее предыдущая версия, поскольку аппаратное обеспечение шилда позволяет использовать только один контакт платы Arduino для управления направлением вращения электродвигателя.

В шилде Ardumoto используются другие контакты, поэтому для работы с ним код скетча нужно откорректировать следующим образом:

```
int leftPins[] = {3, 2}; // Один контакт для ШИМ-сигнала (скорость),
                         // один – для направления
int rightPins[] = {11, 4};
```

В листинге 8.14 приводится код скетча, реализующий эту же функциональность на основе шилда Adafruit Motor Shield V2 (**https://oreil.ly/kFygk**), подключение которого показано на рис. 8.14. В скетче используется библиотека Adafruit_MotorShield, которая устанавливается стандартным способом с помощью Менеджера библиотек.

Шилд Adafruit Motor Shield V2 поддерживает подключение четырех электродвигателей, но в скетче из листинге 8.14 предусмотрено, что два электродвигателя подключены к разъемам шилда 3 и 4.

Листинг 8.14. Управление электродвигателями с помощью шилда Adafruit Motor Shield V2

```
/*
* Скетч Brushed_H_Bridge_Direction sketch for Adafruit Motor shield
* Управляет электродвигателями на основании сигналов фотодатчиков,
* направляя шасси робота к источнику света
*/
```

```
#include <Wire.h>
#include <Adafruit_MotorShield.h> // Подключаем библиотеку Adafruit_MotorShield

// Создаем экземпляр объекта шилда
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *leftMotor = AFMS.getMotor(1);
Adafruit_DCMotor *rightMotor = AFMS.getMotor(2);

const int MIN_PWM = 64;       // Это значение может быть в диапазоне от 0 до MAX_PWM
const int MAX_PWM = 128;      // Это значение может быть в диапазоне от 50 до 255
const int leftSensorPin = A0; // Аналоговые контакты для подключения фотодатчиков
const int rightSensorPin = A1;

int sensorThreshold = 0; // Пороговый уровень освещенности на датчике
                         // для вращения электродвигателя

void setup()
{
   AFMS.begin(); // Инициализируем экземпляр шилда частотой
                 // по умолчанию 1,6 КГц
}

void loop()
{
   int leftVal = analogRead(leftSensorPin);
   int rightVal = analogRead(rightSensorPin);

   if(sensorThreshold == 0) // Датчики откалиброваны?
   {
      // Если нет, калибруем до уровня освещенности несколько выше
      // среднего уровня освещенности окружающей среды
      sensorThreshold = ((leftVal + rightVal) / 2) + 100 ;
   }

   if( leftVal > sensorThreshold || rightVal > sensorThreshold)
   {
     // Если уровень освещенности достаточно высокий для движения вперед
      setSpeed(rightMotor, map(rightVal,0,1023, MIN_PWM, MAX_PWM));
      setSpeed(leftMotor, map(leftVal ,0,1023, MIN_PWM, MAX_PWM));
   }
}

void setSpeed(Adafruit_DCMotor *motor, int speed )
{
   if(speed < 0)
   {
      motor->run(BACKWARD);
```

```
        speed = -speed;
    }
    else
    {
        motor->run(FORWARD);
    }
    motor->setSpeed(speed);
}
```

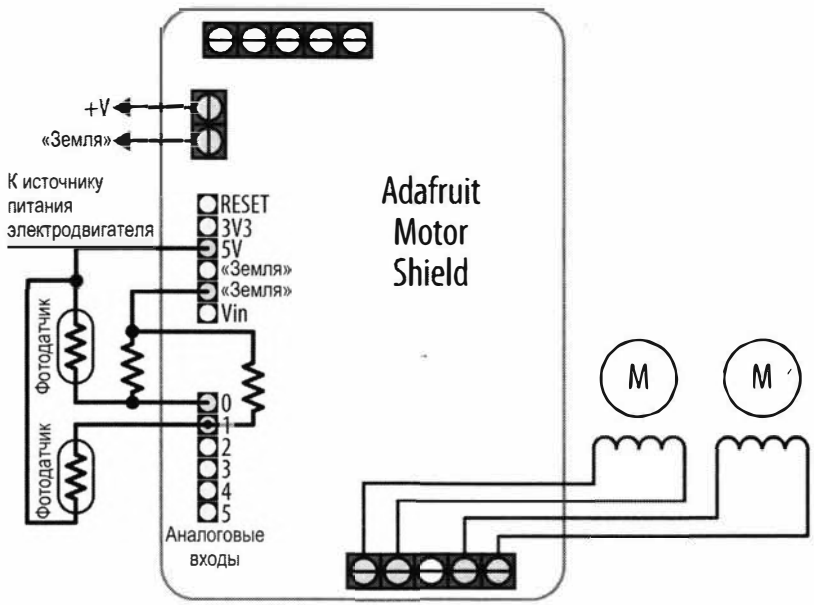


Рис. 8.14. Подключение шилда Adafruit Motor Shield V2

В случае использования иного шилда, чем только что рассмотренные, надо будет уточнить по его справочному листку, что значения в скетче совпадают с контактами, используемыми для сигналов управления скоростью (ШИМ) и направления.

Дополнительная информация

Подробная информация по адаптерной плате компании Pololu для H-моста TB6612FNG приводится в справочном листке на нее (**https://oreil.ly/bD_83**).

Дополнительная информация по шилду Ardumoto компании SparkFun приводится на его веб-странице (**https://oreil.ly/XZTCY**).

Дополнительная информация по шилду Arduino Motor Shield приводится на его веб-странице (**https://oreil.ly/2gKoX**).

Дополнительная информация по шилду Adafruit Motor Shield V2 приводится на его веб-странице (**https://oreil.ly/T19_o**).

8.12. Управление биполярным шаговым двигателем

Требуется программно управлять биполярным (с четырьмя выводами) шаговым электродвигателем через Н-мост.

РЕШЕНИЕ

Для решения этой задачи используется микросхема Н-моста L293, подключение которой к плате Arduino и шаговому электродвигателю показано на рис. 8.15. Конденсаторы номиналом 0,1 мкФ на электродвигателе должны быть керамическими.

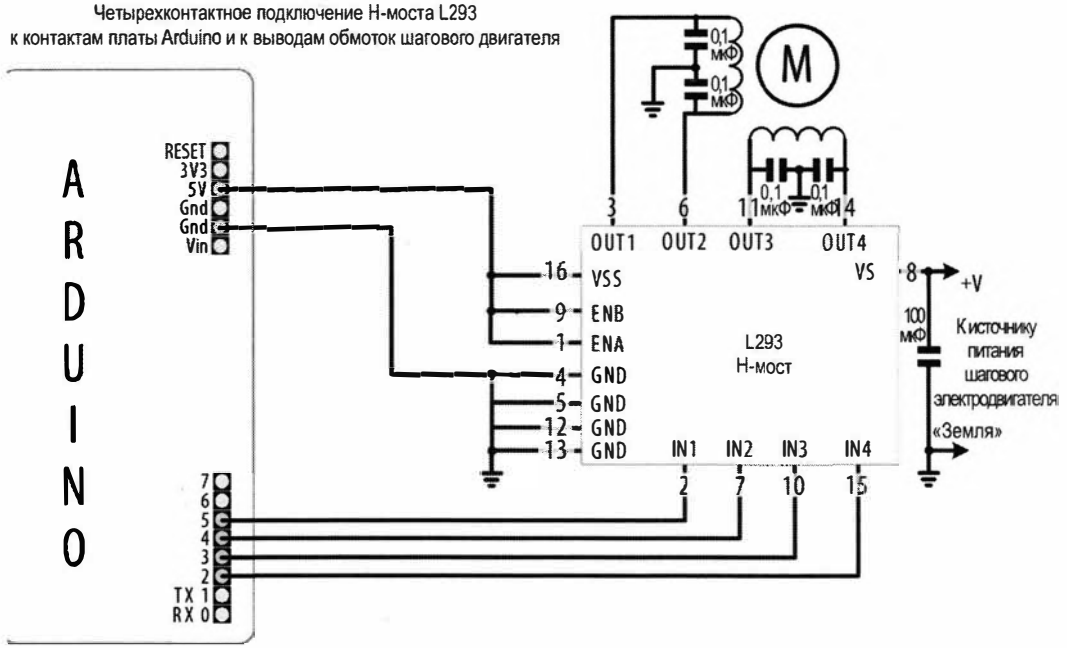


Рис. 8.15. Подключение биполярного шагового электродвигателя к плате Arduino с помощью Н-моста L293

Скетч для управления этой схемой приводится в листинге 8.15. Скетч получает команды для управления электродвигателем и исполняет их. Команда из числового значения со знаком + (плюс) в конце означает пошаговое вращение в одном направлении, а со знаком – (минус) — в другом. Например, команда `24+` заставляет двигатель совершить полный оборот в одном направлении, а команда `12-` — половину оборота в обратном направлении.

Листинг 8.15. Управление шаговым двигателем с помощью Н-моста

```
/*
* Скетч Stepper_bipolar
* Вращает двигатель на число шагов, указанное в командах,
  получаемых по последовательному интерфейсу
```

```
* Команды состоят из числового значения количества шагов
  со знаком направления + или - в конце
*/

#include <Stepper.h>

// Измените это число на количество шагов используемого шагового двигателя
#define STEPS 24

// Создаем экземпляр объекта шагового двигателя, указывая в параметрах
// количество шагов двигателя и контакты платы Arduino для управления им
Stepper stepper(STEPS, 2, 3, 4, 5);

int steps = 0;

void setup()
{
   // Задаем скорость вращения 30 об/мин
   stepper.setSpeed(30);
   Serial.begin(9600);
}

void loop()
{
   if ( Serial.available())
   {
      char ch = Serial.read();
      if(isDigit(ch)) // Значение переменной ch цифра?
      {
         steps = steps * 10 + ch - '0'; // Да, накапливаем значение
      }
      else if(ch == '+')
      {
         stepper.step(steps);
         steps = 0;
      }
      else if(ch == '-')
      {
         stepper.step(steps * -1);
         steps = 0;
      }
   }
}
```

Обсуждение работы решения и возможных проблем

Шаговый двигатель содержит две отдельные группы обмоток, где каждая из групп составляет фазу шагового двигателя. Подача на фазу питания определенной поляр-

ности вызывает вращение электродвигателя на один шаг в соответствующем направлении. Последовательное включение и выключение питания фаз вызывает непрерывное пошаговое вращение двигателя в заданном направлении. Биполярные шаговые двигатели имеют четыре вывода питания — по одной паре для каждой фазы (обмотки). Все эти выводы разного цвета, что позволяет определить по информации в справочном листке, какие из них какой паре обмоток принадлежат. Но иногда цвета в справочном листке не соответствуют цветам выводов на самом устройстве. В таком случае определить выводы обмоток можно простым тестированием их с помощью мультиметра. Для этого нужно измерить сопротивление между парами выводов: те пары выводов, которые имеют одинаковое сопротивление, и будут выводами соответствующих обмоток. Пары выводов с бесконечным сопротивлением будут из разных обмоток.

На шаговый двигатель необходимо подавать правильное напряжение питания и задавать ему правильное количество шагов (`#define STEPS`). Эти характеристики можно узнать из справочного листка или другой документации на конкретный шаговый двигатель. Если используемый шаговый двигатель требует более сильный рабочий ток, чем ток, который может быть предоставлен драйвером L293 (600 мА в случае микросхемы L293D), вместо этого драйвера можно использовать микросхему SN754410, способную предоставить ток силой до 1 ампера. Она подключается так же, как и драйвер L293, и может работать с кодом скетча без каких бы то ни было изменений. А микросхема L298 может предоставить ток силой до 2 ампер. Скетч решения для работы с ней также не требует никаких изменений, но подключается она по-иному, как показано в схеме на рис. 8.16. Конденсаторы номиналом 0,1 мкФ на обмотках электродвигателя должны быть керамическими.

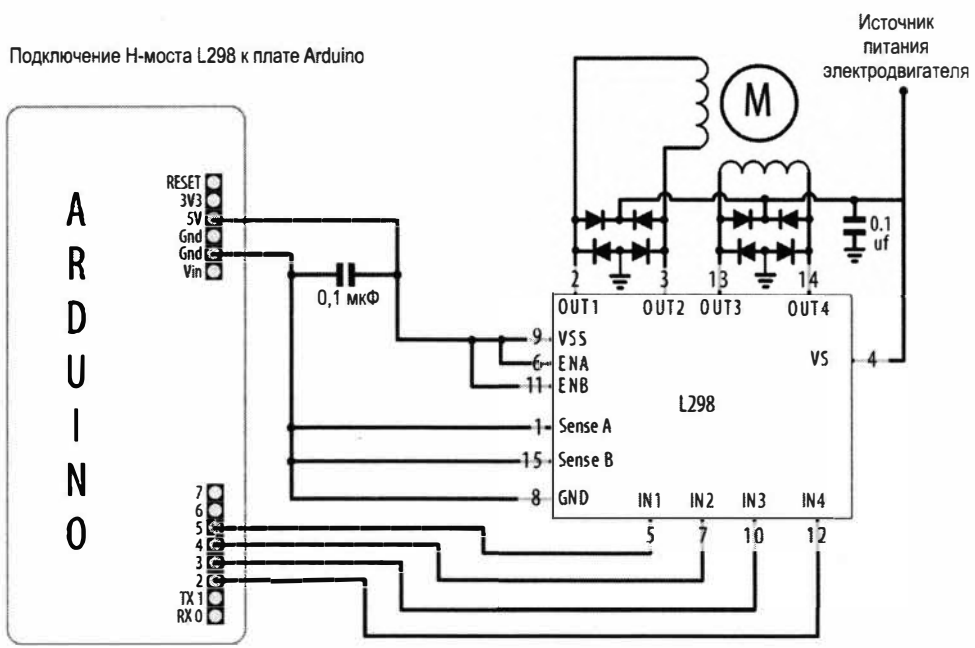


Рис. 8.16. Подключение биполярного шагового электродвигателя к плате Arduino с помощью H-моста L298

Но самый простой способ подключить драйвер L298 к плате Arduino — это использовать шилд Arduino Motor Shield (7630049200371) на основе этой микросхемы. Шилд просто вставляется своими штыревыми разъемами в гнездовые разъемы платы Arduino, и вам останется только подключить к нему выводы обмоток двигателя. Питание для двигателя можно взять с контакта Vin (ввод внешнего напряжения) платы Arduino. Вводы IN1 и IN2 (направление вращения) драйвера управляются контактом 12 платы Arduino, а ввод ENA — контактом 3. Вводы IN3 и IN4 драйвера подключены к контакту 13 платы Arduino, а ввод ENA — к контакту 11. Чтобы использовать этот шилд драйвера, скетч решения нужно модифицировать. В частности, заменить строку кода:

```
Stepper stepper(STEPS, 2, 3, 4, 5);
```

в начале скетча следующим кодом:

```
Stepper stepper(STEPS, 12, 13);
```

А весь код в функции `setup()` заменить кодом из листинга 8.16.

Листинг 8.16. Код функции `setup()` для работы с шилдом H-моста Arduino Motor Shield

```
pinMode(3, OUTPUT);
digitalWrite(3, HIGH);  // Разрешаем двигатель A,
                        // низкий уровень (LOW) выключает двигатель
pinMode(11, OUTPUT);
digitalWrite(11, HIGH); // Разрешаем двигатель B,
                        // низкий уровень (LOW) выключает двигатель
stepper.setSpeed(60);   // Задаем скорость вращения 60 об/мин
Serial.begin(9600);
```

Остальной код такой же, как и в скетче решения (см. листинг 8.15).

Шаговые двигатели потребляют значительный ток, даже если не вращаются. Попробуйте слегка провернуть (именно только попробуйте и только слегка, а не в самом деле проверните) вал не вращающегося, но запитанного шагового двигателя, и вы должны почувствовать сопротивление вашему усилию. После определенного времени работы установленная на макетной плате микросхема драйвера L293 и даже более мощная микросхема L298 нагреется до очень высокой температуры, что может вызвать плавление пластика макетной платы. По этой причине настоятельно рекомендуется использовать рассматриваемый далее шилд EasyDriver драйвера шаговых двигателей. Микросхемы драйверов шилдов шаговых двигателей обычно оснащаются теплоотводом, что повышает их термостойкость при таком использовании. С целью экономии электроэнергии питание шагового двигателя можно отключать, когда он не вращается. Для этого нужно подать сигнал низкого уровня на контакты ENA и ENB драйвера. Но это часто идет вразрез с целью использования шагового двигателя, т. е. возможностью удерживания позиции при отсутствии вращения. Компромиссное решение в этом отношении заключается в отключении питания шагового двигателя после определенного периода отсутствия вращения. В *разд. 8.13* рассматривается пример такого решения.

Дополнительная информация

Дополнительная информация по подключению шаговых двигателей приводится в заметках Тома Иго (Tom Igoe) по шаговым двигателям (**https://oreil.ly/-Phfq**).

С документацией по библиотеке Stepper можно ознакомиться на ее веб-странице (**https://oreil.ly/PVcIJ**).

Дополнительная информация по шаговым двигателям приводится в соответствующем руководстве компании Adafruit (**https://oreil.ly/AJaAG**).

8.13. Управление биполярным шаговым двигателем с использоваением платы EasyDriver

ЗАДАЧА

Требуется программно управлять биполярным (с четырьмя выводами) шаговым электродвигателем, используя плату EasyDriver компании Adafruit (артикул ROB-12779, **https://www.sparkfun.com/products/12779**).

РЕШЕНИЕ

Решение этой задачи похоже на решение задачи из *разд. 8.12* и использует ту же систему передачи команд по последовательному интерфейсу, но вместо драйвера L293/L298 в нем задействована микросхема драйвера A3967, установленная на адаптерной плате EasyDriver компании Adafruit. Подключение этой адаптерной платы и шагового двигателя к плате Arduino показано на рис. 8.17.

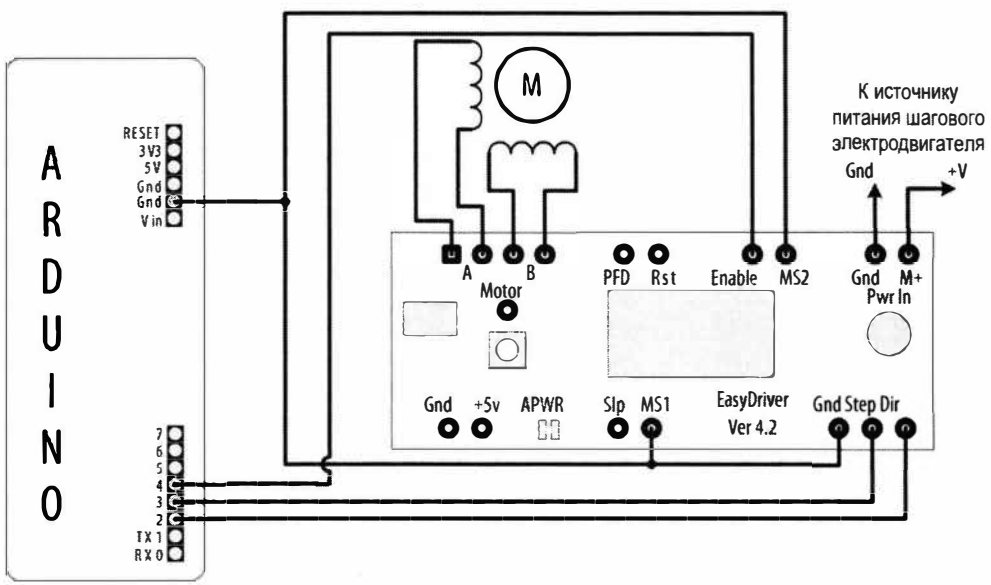


Рис. 8.17. Подключение шагового двигателя к плате Arduino с помощью адаптерной платы H-моста EasyDriver

Скетч из листинга 8.17 управляет направлением вращения и количеством шагов двигателя на основе команд, получаемых через последовательный интерфейс. В отличие от скетча из *разд. 8.12*, для этого скетча не требуется библиотека Stepper, поскольку управление подачей питания на обмотки двигателя выполняется аппаратно платой EasyDriver.

Листинг 8.17. Управление биполярным шаговым двигателем с помощью платы EasyDriver

```
/*
* Скетч Stepper_Easystepper
* Исполняет команды для управления шаговым двигателем,
  получаемые по последовательному интерфейсу
* Команды состоят из числового значения количества шагов
  со знаком направления + или - в конце
* Команды из числового значения с символом "s" в конце меняют скорость
*/

const int dirPin = 2;
const int stepPin = 3;
const int enPin = 4;

int speed = 100; // Требуемая скорость в шагах в секунду
int steps = 0;   // Требуемое количество шагов

long last_step = millis();
long timeout = 30 * 1000; // Выключаем питание двигателя
                          // после 30 секунд отсутствия вращения

void setup()
{
   pinMode(dirPin, OUTPUT);
   pinMode(stepPin, OUTPUT);
   pinMode(enPin, OUTPUT);
   Serial.begin(9600);
}

void loop()
{
   if (millis() > last_step + timeout)
   {
      digitalWrite(enPin,HIGH); // Выключаем электродвигатель
   }

   if ( Serial.available())
   {
      char ch = Serial.read();
```

```
      if(isDigit(ch)) // Значение переменной ch цифра?
      {
         steps = steps * 10 + ch - '0'; // Да, накапливаем значение
      }
      else if(ch == '+')
      {
         step(steps);
         steps = 0;
      }
      else if(ch == '-')
      {
         step(-steps);
         steps = 0;
      }
      else if(ch == 's')
      {
         speed = steps;
         Serial.print("Setting speed to "); // Задаем скорость
         Serial.println(steps);
         steps = 0;
      }
   }
}

void step(int steps)
{
   int stepDelay = 1000 / speed; // Пауза в миллисекундах для скорости,
                                 // заданной в шагах в секунду
   int stepsLeft;

   digitalWrite(enPin,LOW); // Разрешаем двигатель
   last_step = millis();

   // Определяем направление вращения по знаку переменной steps: + или -
   if (steps > 0)
   {
      digitalWrite(dirPin, HIGH);
      stepsLeft = steps;
   }
   if (steps < 0)
   {
      digitalWrite(dirPin, LOW);
      stepsLeft = -steps;
   }
   // Декрементируем количество шагов, вращая двигатель по шагу каждый раз
   while(stepsLeft > 0)
   {
      digitalWrite(stepPin,HIGH);
```

```
      delayMicroseconds(1);
      digitalWrite(stepPin,LOW);
      delay(stepDelay);
      stepsLeft--; // Декрементируем оставшееся количество шагов
   }
}
```

Обсуждение работы решения и возможных проблем

Питание для шагового двигателя подается на контакты M+ и Gnd платы EasyDriver (в правом верхнем углу схемы на рис. 8.17). Плата поддерживает напряжения от 8 до 30 вольт. Проверьте требуемое рабочее напряжение вашего шагового двигателя в технической документации на него. В случае использования шагового двигателя с рабочим напряжением 5 В, на контакты платы, обозначенные Gnd и +5v (в левом нижнем углу платы EasyDriver), необходимо подать питание напряжением 5 В. Кроме того, надо также перерезать обозначенную символами APWR перемычку на печатной плате справа от этих контактов. Так будет отключен встроенный стабилизатор напряжения платы, и шаговый двигатель, и сама плата станут получать питание с внешнего источника питания напряжением 5 В.

Скетч решения содержит энергосберегающую функцию, отключая питание шагового двигателя при отсутствии вращения свыше 30 секунд подачей сигнала высокого уровня (`HIGH`) на контакт Enable платы EasyDriver (включается двигатель подачей на этот контакт сигнала низкого уровня `LOW`). Величину периода тайм-аута можно откорректировать, изменив значение переменной `last_step`.

Тип шага (полный шаг или один из микрошагов) задается комбинациями высокого (`HIGH`) и низкого (`LOW`) уровней на контактах MS1 и MS2 платы, подключая эти контакты к контакту 5V или Gnd платы Arduino соответственно, как показано в табл. 8.2. Подача на оба эти контакта низкого уровня, как показано ранее на рис. 8.17, устанавливает полношаговый режим работы двигателя. Обратите также внимание, что уровнем по умолчанию на контакте сброса Rst является высокий (отсутствие подключения к шине «земли»). Установка на этом контакте низкого уровня отключает управление шаговым двигателем.

***Таблица 8.2.** Опции шаговых режимов*

Разрешение шага	MS1	MS2
Полный шаг	Низкий (`LOW`)	Низкий (`LOW`)
Полшага	Высокий (`HIGH`)	Низкий (`LOW`)
Четверть шага	Низкий (`LOW`)	Высокий (`HIGH`)
Одна восьмая шага	Высокий (`HIGH`)	Высокий (`HIGH`)

Код скетча можно модифицировать с тем, чтобы количество оборотов в секунду определялось значением переменной speed:

```
// Используйте следующий код для скорости, задаваемой в об/мин
int speed = 100;              // Требуемая скорость в об/мин
int stepsPerRevolution = 200; // Количество шагов на один оборот
```

В функции step() замените первую строку кода следующей:

```
// Скорость в об/мин
int stepDelay = 60L * 1000L / stepsPerRevolution / speed;
```

Остальной код остается без изменений, но теперь команда скорости будет задавать не количество шагов в секунду, а количество оборотов в минуту.

8.14. Управление униполярным шаговым двигателем с помощью драйвера ULN2003A

ЗАДАЧА

Требуется управлять униполярным (с пятью или шестью выводами) шаговым двигателем с помощью микросхемы драйвера на парах Дарлингтона ULN2003A.

РЕШЕНИЕ

Подключение микросхемы драйвера к плате Arduino и шаговому электродвигателю показано на рис. 8.18. Контакт +V микросхемы подключается к источнику питания с напряжением и током, достаточными для используемого шагового двигателя. Конденсатор номиналом 0,1 мкФ на линии питания электродвигателя должен быть керамическим.

Скетч из листинга 8.18 управляет направлением вращения и количеством шагов двигателя на основе команд, получаемых через последовательный интерфейс. Команда из числового значения со знаком + (плюс) в конце означает пошаговое вращение в одном направлении, а со знаком - (минус) — в другом.

Листинг 8.18. Управление униполярным шаговым электродвигателем

```
 /*
* Скетч Stepper
* Исполняет команды для управления шаговым двигателем,
  получаемые по последовательному интерфейсу
* Команды состоят из числового значения количества шагов
  со знаком направления + или - в конце
*/

#include <Stepper.h>
```

```
// Измените это число на количество шагов
// используемого шагового двигателя
#define STEPS 24

// Создаем экземпляр объекта шагового двигателя, указывая
// в параметрах количество шагов двигателя и контакты платы Arduino для управления им
Stepper stepper(STEPS, 2, 3, 4, 5);

int steps = 0;

void setup()
{
   stepper.setSpeed(30); // Задаем скорость вращения 30 об/мин
   Serial.begin(9600);
}

void loop()
{
   if ( Serial.available())
   {
      char ch = Serial.read();

      if(isDigit(ch)) // Значение переменной ch цифра?
      {
         steps = steps * 10 + ch - '0'; // Да, накапливаем значение
      }
      else if(ch == '+')
      {
         stepper.step(steps);
         steps = 0;
      }
      else if(ch == '-')
      {
         stepper.step(steps * -1);
         steps = 0;
      }
      else if(ch == 's')
      {
         stepper.setSpeed(steps);
         Serial.print("Setting speed to "); // Задаем скорость в
         Serial.println(steps);
         steps = 0;
      }
   }
}
```

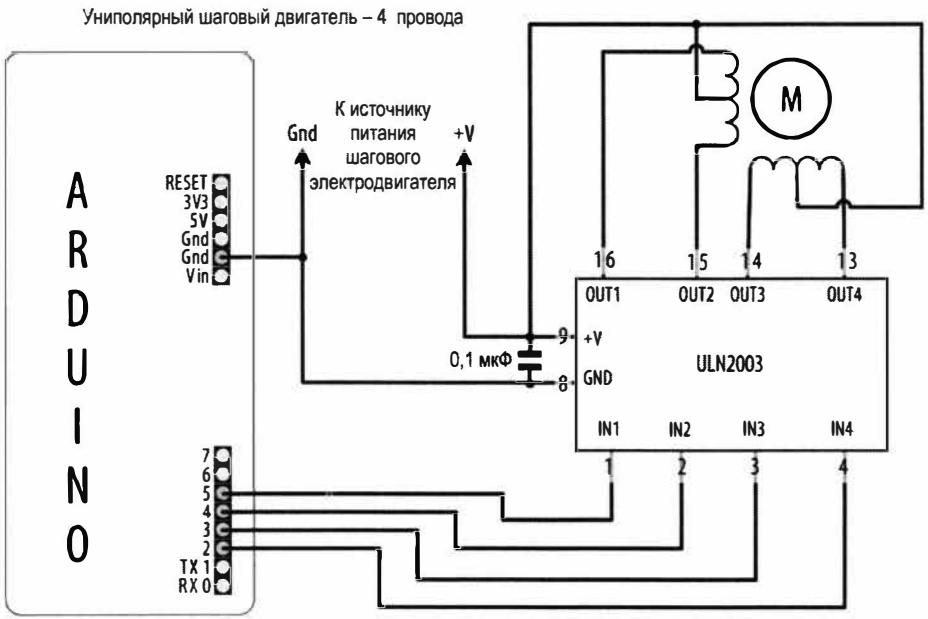


Рис. 8.18. Подключение униполярного шагового электродвигателя к плате Arduino с помощью драйвера ULN2003

Обсуждение работы решения и возможных проблем

Униполярные шаговые электродвигатели, так же, как и биполярные, тоже имеют две обмотки, но с середины каждой обмотки делается дополнительный вывод. Эти средние выводы могут внутри двигателя соединяться вместе, и тогда наружу выходит только один общий вывод. В результате эти двигатели имеют пять выводов. Определить, какие выводы к каким обмоткам относятся, можно с помощью мультиметра, измеряя сопротивление между парами выводов. Пара выводов концов обмотки будет иметь самое большое сопротивление. Таких пар выводов должно быть две — с более или менее одинаковым сопротивлением. Затем определяется центральный вывод для каждой обмотки — измерением сопротивления между оставшимися двумя выводами и выводами концов обмотки. Сопротивление между выводом конца обмотки и выводом от ее середины должно быть вдвое меньшим сопротивления всей обмотки. На веб-странице **https://oreil.ly/7ckQc** приводится пошаговая процедура (на английском языке) определения выводов обмоток шаговых двигателей.

Дополнительная информация

Дополнительная информация по подключению шаговых двигателей приводится в заметках Тома Иго (Tom Igo) по шаговым двигателям (**http://www.tigoe.net/pcomp/code/circuits/motors**).

С документацией по библиотеке Stepper можно ознакомиться на ее веб-странице (**https://oreil.ly/-Phfq**).

ГЛАВА 9

Работа со звуком

9.0. Введение

Поиск в Google по ключевой фразе типа `музыкальный проект на Arduino` возвращает бесчисленное множество результатов, но в этой главе у нас есть достаточно места только для того, чтобы дать введение в методы работы со звуком с использованием платформы Arduino и привести несколько примеров в качестве отправной точки для работы в этой области. Плата Arduino не предназначена для использования в качестве продвинутого синтезатора звука, но она может воспроизводить звуки определенного качества через какое-либо устройство вывода — например, динамик.

В этой главе мы рассмотрим способы создания звуковых эффектов и методы простого аудиовывода для воспроизведения записанных звуков, а также немного поэкспериментируем с синтезированием звука.

Если вы испытываете затруднения с созданием для своего аудиопроекта, в Сети есть много веб-сайтов музыкальных проектов, посещение которых может помочь вам решить эту проблему. Вот, например, следующие два:

- страница блога на веб-сайте Arduino, содержащая 187 музыкальных проектов (**https://oreil.ly/3Qbw5**);
- еще одна страница блога на веб-сайте Arduino, содержащая более продвинутые музыкальные проекты (**https://oreil.ly/q60bC**).

А следующие сайты — это всего лишь два примера того, что можно сделать с помощью Arduino, небольшого количества разных деталей и запаса творческой энергии:

- «Лазерная арфа» (**https://oreil.ly/ZCAIH**);
- «Терменвокс» (**https://oreil.ly/KgyaX**). Кстати, в *разд. 9.6* рассматривается очень упрощенная версия этого электронного музыкального инструмента, однако на этом сайте приводится информация для создания его настоящего воплощения.

Звук возникает, когда что-либо заставляет вибрировать воздух. Регулярно повторяющиеся вибрации генерируют звук определенной тональности. С помощью платы Arduino можно создавать звуки, подавая сигнал на динамический или пьезоэлектрический громкоговоритель (далее в тексте — динамик), превращающий электронные импульсы в пульсации мембраны, которые, в свою очередь, создают вибрации воздуха, которые мы воспринимаем как звук. Тембр (частота) звука опре-

деляется длительностью периода времени, в течение которого мембрана динамика совершает одно полное колебание. Чем короче этот период времени, тем выше частота создаваемого звука.

Наиболее распространены два типа пьезоэлектрических динамиков (керамических преобразователей небольшого размера, создающих звук при подаче на них импульсов напряжения). *Пьезоэлектрический громкоговоритель* может создавать звуки в определенном диапазоне частот. С другой стороны, устройство, называемое *пьезозуммером*, содержит схему генератора колебаний, который при подаче на зуммер питания создает звук постоянной частоты. В проектах с использованием пьезоустройств в этой главе предполагается использование пьезоэлектрического громкоговорителя, а не пьезозуммера.

Частота колебаний означает количество полных колебаний, которые совершает сигнал в течение одной секунды, и измеряется в герцах. Человеческий слух может воспринимать звуки с частотой колебаний в диапазоне от приблизительно 20 герц (Гц) до приблизительно 20 тысяч герц (хотя этот диапазон зависит от конкретного человека и его возраста).

Программное обеспечение Arduino содержит функцию `tone()`, обеспечивающую генерацию звуковых импульсов. В *разд. 9.1* и *9.2* рассматривается применение этой функции для создания простых звуков и мелодий. Функция `tone()` использует в своей работе аппаратные таймеры. Стандартные платы Arduino (Uno и подобные ей) поддерживают одновременное воспроизведение тона только одной частоты. В процессе своей работы функция `tone()` занимает таймер, используемый для функции `analogWrite()` на контактах 3 и 11, поэтому, если вам требуется аналоговый вывод, придется задействовать другие контакты. В *разд. 9.3* показано, как решить эту проблему с помощью расширенной библиотеки Tones, помогающей создавать одновременно звуки разной тональности, а в *разд. 9.4* рассказано, как создавать звуки, не прибегая к использованию функции `tone()` и аппаратных таймеров.

Подача на громкоговоритель последовательности импульсов генерирует звук ограниченного качества, который мало напоминает музыкальные звуки. В частности, звук, создаваемый сигналом прямоугольной формы (рис. 9.1), звучит жестко и более похож на звук из старых компьютерных игр, чем на звучание музыкального инструмента.

Простые платы (наподобие Arduino Uno) плохо подходят для генерирования более сложных музыкальных звуков без использования дополнительного оборудования. Расширить возможности платы Arduino Uno в этом плане можно с помощью специального шилда — например, шилда Wave Shield компании Adafruit, который позволяет воспроизводить аудиофайлы, хранящиеся на карте памяти шилда.

Некоторые из более новых плат Arduino оснащены встроенным цифроаналоговым преобразователем, который может генерировать качественный звук, воспроизводя звуковые файлы, хранящиеся на SD-картах, или синтезируя звук программным способом (см. *разд. 1.8*). Другой возможностью более новых плат является наличие в них интерфейса I^2S (Inter-IC Sound, шина для соединения аудиоустройств). Это цифровой интерфейс для взаимодействия с внешними микросхемами, реали-

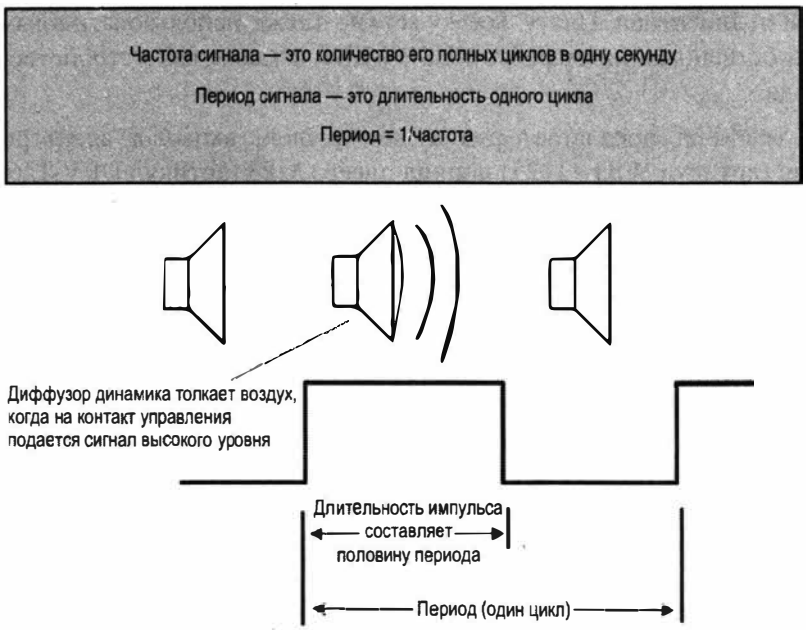


Рис. 9.1. Генерирование звуков с помощью цифровых импульсов

зующий высококачественный стереофонический звуковой интерфейс, как входной, так и выходной.

Плату Arduino можно также использовать для управления внешними устройствами, предназначенными для генерирования звуков. В *разд. 9.5* рассматривается решение для отправки MIDI-сообщений (Musical Instrument Digital Interface, цифровой интерфейс музыкальных инструментов) на устройство MIDI. Устройства MIDI могут синтезировать высококачественные звуки, воспроизводящие звучание обширнейшего круга музыкальных инструментов, и способны одновременно создавать звуки нескольких инструментов. В *разд. 9.5* показано, как генерировать сообщения MIDI для воспроизведения музыкальной гаммы.

В *разд. 9.6* рассматривается приложение Auduino, которое использует для синтезирования звука сложную программную технологию. И наконец, в *разд. 9.7* исследуется библиотека для синтезирования звука с более продвинутыми возможностями.

Для экспериментирования с продвинутыми приложениями для генерирования звука хорошо подойдут 32-разрядные платы Teensy компании PJRC (**https://www.pjrc.com/teensy/index.html**). При работе с этой платой можно задействовать продвинутую библиотеку (**https://oreil.ly/GBTYg**), которая реализует возможности цифровой обработки сигналов микроконтроллера платы Teensy для синтезирования сложных звуковых эффектов посредством встроенного ЦАП, позволяя получить настоящий аналоговый аудиовывод. Для этих плат предлагается аудиошилд (**https://oreil.ly/qRMCZ**), оснащенный встроенным считывателем SD-карт и микросхемой с интерфейсом I^2S, которая позволяет генерировать выходной 16-битный стереосигнал частотой 44,1 КГц, а также способна обрабатывать входящий стерео-

фонический аудиосигнал. Плату Teensy можно также использовать в качестве устройства с собственным интерфейсом USB-MIDI, а также как устройство звукового ввода/вывода.

Компания SparkFun предлагает ряд аудиомодулей, включая адаптерную плату Audio-Sound (артикул WIG-11125) и шилд плеера MP3 (артикул DEV-12660).

Платы Arduino Zero, MKR1000 и MKRZero также оснащены ЦАП, что позволяет воспроизводить файлы WAV, сохраненные на SD-карте. Соответствующие рекомендации приводятся на веб-странице **https://oreil.ly/lG0as**.

В этой главе рассматриваются различные способы генерирования звуков с помощью электронных схем. Кроме того, устройства, использующие платформу Arduino, способны играть и на обычных музыкальных инструментах — например, металлофоне, барабанах или пианино. Для этого надо будет только задействовать силовые приводы — такие как соленоиды и сервомашинки, рассматриваемые в *главе 8*.

Во многих решениях этой главы выходной сигнал подается на небольшой динамик или пьезоэлектрический громкоговоритель. Схема для подключения таких выходных аудиоустройств показана на рис. 9.2.

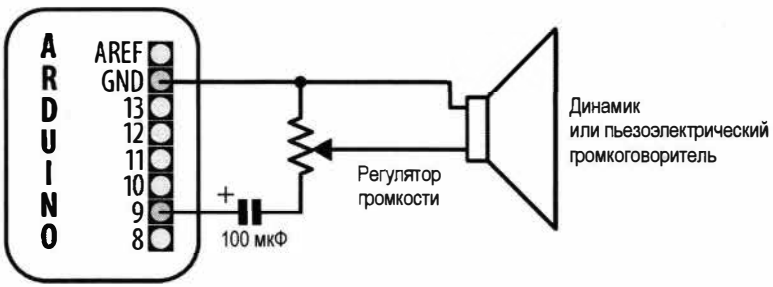


Рис. 9.2. Подключение динамика или пьезоэлектрического громкоговорителя к плате Arduino

Для регулирования громкости воспроизводимого сигнала используется потенциометр номиналом от 200 до 500 Ом. Электролитный конденсатор емкостью 100 мкФ подключается положительным выводом к контакту Arduino. Обычный динамик можно подключать к «земле» любым выводом, но выводы пьезоэлектрического громкоговорителя поляризованы, и к контакту «земли» (GND) платы Arduino следует подключать отрицательный вывод (который обычного черного цвета).

Выходной сигнал можно также сначала подавать на внешний усилитель низкой частоты (УНЧ), а динамик подключать уже к выходу этого усилителя. Подключение выходного сигнала к штыревому разъему (штекеру) усилителя показано на рис. 9.6.

При воспроизведении выходного аудиосигнала на наушники, прежде чем надевать их, убедитесь, что установлен безопасный уровень громкости. В зависимости от используемой платы и способа подключения наушников выходной аудиосигнал может быть весьма громким.

9.1. Воспроизведение звуков разной частоты

ЗАДАЧА

Требуется воспроизводить звуки разной частоты и длительности с помощью динамика.

РЕШЕНИЕ

Эта задача решается с помощью функции `tone()`. Схема для подключения динамика и потенциометров для управления частотой и длительностью звука показана на рис. 9.3, а в листинге 9.1 приводится скетч для работы с этой схемой.

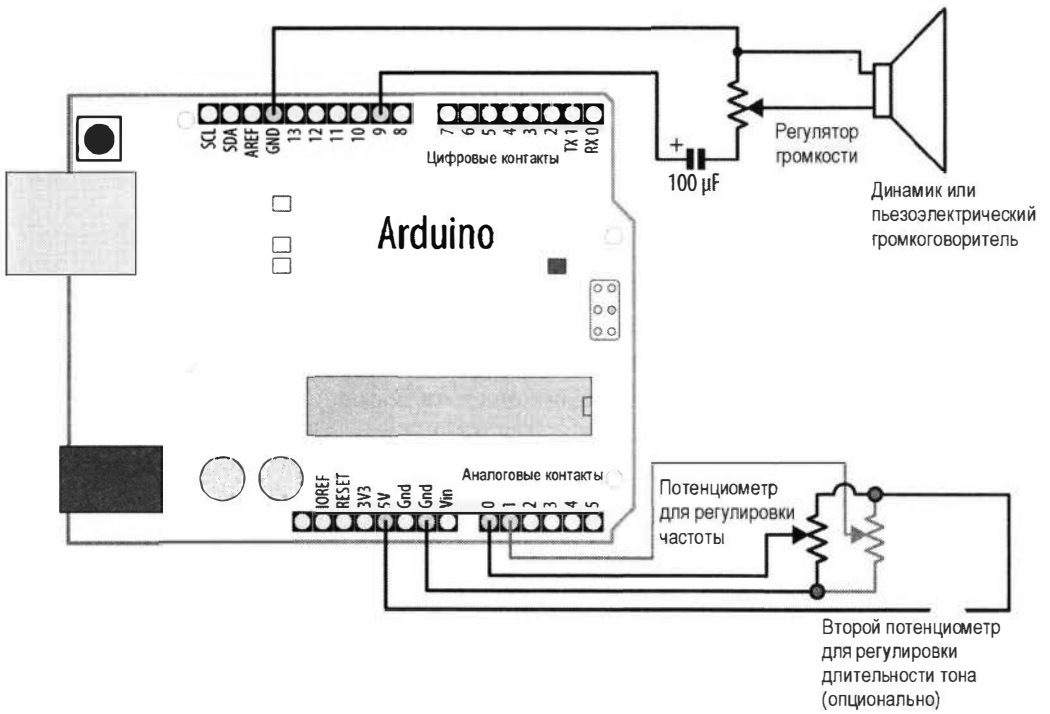


Рис. 9.3. Подключение динамика для воспроизведения тонов разной частоты и длительности (последнее опционально)

Листинг 9.1. Скетч для воспроизведения звуков разной частоты

```
/*
* Скетч Tone
*
* Подает звуковой сигнал на контакт 9 для воспроизведения на динамике
* Частота тона задается значением сигнала потенциометра,
  считываемого с входного аналогового контакта
*/
```

```
const int speakerPin = 9; // Контакт для подключения динамика
const int pitchPin = A0;  // Контакт для подключения потенциометра,
                          // определяющего частоту тона

void setup()
{
   // Ничего не делаем здесь.
}

void loop()
{
   int sensor0Reading = analogRead(pitchPin); // Считываем сигнал
                          // потенциометра для установки частоты

   // Масштабируем считанный сигнал к значимому диапазону
   int frequency = map(sensor0Reading, 0, 1023, 100, 5000); // Выходной
                                        // диапазон: от 100 Гц до 5 КГц

   int duration = 250; // Длительность воспроизведения тона
   tone(speakerPin, frequency, duration); // Воспроизводим тон
   delay(1000); // Выдерживаем паузу в одну секунду
}
```

Функции `tone()` может передаваться до трех параметров: номер контакта платы Arduino, к которому подключен динамик, частота воспроизводимого тона в герцах и длительность его воспроизведения в миллисекундах. Первые два параметра являются обязательными, а третий — опциональным. При отсутствии третьего параметра тон будет воспроизводиться до следующего вызова функции `tone()` или `noTone()`. Значения сигнала потенциометра, задающие частоту воспроизводимого тона, масштабируются до значимого диапазона значений звуковых частот следующей строкой кода:

```
int frequency = map(sensor0Reading, 0, 1023, 100, 5000); // Выходной
                                                  // диапазон: от 100 Гц до 5 КГц
```

В листинге 9.2 приводится версия скетча, задействующая второй потенциометр для задания длительности воспроизведения тона (см. рис. 9.3, *справа внизу*).

Листинг 9.2. Версия скетча для воспроизведения тонов разной частоты и длительности

```
const int speakerPin = 9;    // Контакт для подключения динамика
const int pitchPin = A0;     // Контакт для подключения потенциометра
                             // управления частотой тона
const int durationPin = A1; // Контакт для подключения потенциометра
                             // управления длительностью тона
```

```
void setup()
{
   // Ничего не делаем здесь
}

void loop()
{
   int sensor0Reading = analogRead(pitchPin);     // Считываем сигнал
                        // потенциометра для установки частоты тона
   int sensor1Reading = analogRead(durationPin); // Считываем сигнал
                    // потенциометра для установки длительности тона

   // Масштабируем считанные сигналы к значимому диапазону
                        // Частота: от 100 Гц до 5 КГц
   int frequency = map(sensor0Reading, 0, 1023, 100, 5000);
                        // Длительность: от 0,1 до 1 секунды
   int duration = map(sensor1Reading, 0, 1023, 100, 1000);
   tone(speakerPin, frequency, duration); // Воспроизводим тон
   delay(duration); // Ждем окончания воспроизведения
}
```

Решение можно дальше модифицировать, добавив в схему кнопку, чтобы тон воспроизводился только при ее нажатии. Для этого нужно доработать скетч, указав в нем номер контакта `inputPin` для подключения кнопки и задав для него входной режим работы, используя встроенный повышающий резистор:

```
pinMode(inputPin, INPUT_PULLUP);
```

Кроме того, код главного цикла `loop()` нужно подправить, чтобы функции `tone()` и `delay()` вызывались только при нажатой кнопке:

```
if (digitalRead(inputPin) == LOW)  // Считываем входное значение состояния кнопки
{
   tone(speakerPin, frequency, duration); // Воспроизводим тон
   delay(duration); // Ждем окончания воспроизведения
}
```

Для воспроизведения звуков совместно с платой Arduino можно задействовать практически любой электроакустический преобразователь. Прежде всего, хорошо подходят небольшие динамики. Вполне пригодны также пьезоэлектрические громкоговорители — недорогие и надежные устройства, которые можно извлечь из старых звуковоспроизводящих поздравительных открыток. Пьезодинамики имеют высокое входное сопротивление, вследствие чего потребляют очень малый ток, что позволяет подключать их непосредственно к управляющему контакту платы Arduino. У стандартных динамиков обычно намного более низкое входное сопротивление, вследствие чего они требуют подключения последовательного токоограничивающего резистора. Найти все компоненты, необходимые для сборки схемы, приведенной на рис. 9.3, не должно представлять какой-либо большой трудности.

Дополнительная информация

Функциональность скетча можно расширить, используя библиотеку Tone Бретта Хагмана (Brett Hagman), рассматриваемую в *разд. 9.3*.

9.2. Проигрывание простой мелодии

ЗАДАЧА

Воспроизвести с помощью платы Arduino простую мелодию.

РЕШЕНИЕ

Для решения этой задачи можно использовать функцию `tone()`, рассмотренную в *разд. 9.1*, позволяющую воспроизводить на динамике музыкальные ноты. Схему соединений компонентов можно использовать ту же, что и для решения в *разд. 9.1*. А в листинге 9.3 приводится код скетча, который воспроизводит музыкальный эквивалент программы «Здравствуй, мир» при обучении игре на пианино — последовательность нот мелодии «Twinkle, Twinkle Little Star» («Мерцай, мерцай, звездочка»).

Листинг 9.3. Скетч для воспроизведения мелодии «Twinkle, Twinkle Little Star»

```
/*
* Скетч Twinkle
* Воспроизводит мелодию "Twinkle, Twinkle Little Star"
* Динамик подключается к контакту 9 платы Arduino
*/

const int speakerPin = 9; // Контакт для подключения динамика

char noteNames[] = {'C','D','E','F','G','a','b'};
unsigned int frequencies[] = {262,294,330,349,392,440,494};
const byte noteCount = sizeof(noteNames); // Количество нот (7 в нашем случае)

// Ноты мелодии, пробел обозначает отсутствие звука
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
const byte scoreLen = sizeof(score); // Количество в мелодии

void setup()
{
   // Здесь ничего не делаем
}

void loop()
{
```

```
  for (int i = 0; i < scoreLen; i++)
  {
    int duration = 333; // Каждая нота длится треть секунды
    playNote(score[i], duration); // Воспроизводим ноту
    delay(duration/10); // Небольшая пауза для разделения нот
  }
  delay(4000); // Пауза в четыре секунды перед повторным
               // воспроизведением мелодии
}

void playNote(char note, int duration)
{
  // Воспроизводим тон, соответствующий названию ноты
  for (int i = 0; i < noteCount; i++)
  {
    // Пробуем найти совпадения для значения noteName,
    // чтобы получить указатель на ноту
    if (noteNames[i] == note) // Находим соответствующую ноту в массиве
    tone(speakerPin, frequencies[i], duration); // Воспроизводим ноту
  }
  // Если нет совпадения, тогда нотой является пауза, которую и выдерживаем
  delay(duration);
}
```

Массив символов `noteNames` используется для идентификации нот мелодии. Каждый элемент этого массива сопоставляется частоте, определенной в массиве `notes`. Например, нота `C` (первый элемент массива `noteNames`) имеет частоту 262 Гц (первый элемент в массиве `notes`).

Массив нот `score` содержит названия нот мелодии, которую нужно воспроизвести, — названия нот, записанные прописными буквами (`CCGG` и т. п.), на октаву ниже названий нот, записанных строчными буквами (`aa`):

```
// Ноты мелодии, пробел обозначает отсутствие звука
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
```

Каждый символ мелодии, который совпадает с символом в массиве `noteNames`, вызывает воспроизведение соответствующей ноты. Символы пробела означают паузу в воспроизведении, но любой символ, не определенный в массиве `noteNames`, также вызовет паузу.

Для каждого символа последовательности нот мелодии вызывается функция `playNote()`, которая и воспроизводит эту ноту в течение трети секунды.

Функции `playNote()` выполняет поиск в массиве `noteNames` символа, совпадающего с текущим символом нотной последовательности, и использует соответствующий элемент в массиве частот, чтобы получить частоту для воспроизведения.

Все ноты имеют одинаковую длительность. При желании для каждой ноты можно указывать конкретную длительность, добавив в скетч следующий код:

```
byte beats[scoreLen] = {1,1,1,1,1,1,2, 1,1,1,1,1,1,2,1,
                        1,1,1,1,1,1,2, 1,1,1,1,1,1,2,1,
                        1,1,1,1,1,1,2, 1,1,1,1,1,1,2};
byte beat = 180;                    // Тактов в минуту для восьмых нот
unsigned int speed = 60000 / beat; // Время одного такта в миллисекундах
```

Каждый элемент массива `beats` содержит тип соответствующей ноты мелодии: значение `1` означает восьмую ноту, `2` — четвертную, и т. д.

Переменная `beat` содержит значение количества тактов в минуту.

Переменная `speed` содержит результат вычисления для преобразования количества тактов в минуту в миллисекунды.

В коде главного цикла `loop()` нужно внести только одно изменение, чтобы переменной длительности ноты `duration` присваивалось значение из массива `beats`. В частности, следующую строку кода:

```
int duration = 333; // Каждая нота длится треть секунды
```

нужно заменить этой:

```
int duration = beats[i] * speed; // Длительность звучания каждой ноты
    // определяется соответствующим значением элемента массива beats
```

9.3. Генерирование несколько тонов одновременно

ЗАДАЧА

Требуется генерировать одновременно два звука. Библиотека Tone среды Arduino IDE поддерживает воспроизведение только одного тона на стандартных платах, но мы хотим создавать одновременно два тона. При этом следует отметить, что плата Mega оснащена большим количеством таймеров и способна воспроизводить до шести тонов одновременно.

РЕШЕНИЕ

Возможности библиотеки Tone среды Arduino IDE ограничены одновременным воспроизведением лишь одного тона. Это объясняется тем, что для каждого тона требуется использовать отдельный таймер, и хотя стандартная плата Arduino оснащена тремя таймерами, один из них занят функцией `millis()`, а еще один требуется для работы с сервомашинками. Чтобы решить эту проблему, здесь применена библиотека Tone с расширенными возможностями разработки Бретта Хагмана (Brett Hagman), который также является и разработчиком функции `tone()`. Его библиотека позволяет генерировать одновременно несколько тонов. Библиотеку можно загрузить с ее веб-страницы Github (**https://github.com/bhagman/Tone**) или же просто установить с помощью Менеджера библиотек.

В листинге 9.4 приводится код скетча, который проигрывает часть мелодии «Twinkle, Twinkle Little Star» с теми же самыми нотами, но в двух октавах.

Листинг 9.4. Скетч для воспроизведения нескольких тонов одновременно на стандартной плате Arduino

```
/*
* Скетч Dual Tones
* Проигрывает фрагмент мелодии "Twinkle, Twinkle Little Star" в двух октавах
*/

#include <Tone.h>

int notes1[] = {NOTE_C3, NOTE_C3, NOTE_G3, NOTE_G3, NOTE_A4, NOTE_A4,
                NOTE_G3, NOTE_F3, NOTE_F3, NOTE_E3, NOTE_E3, NOTE_D3,
                NOTE_D3, NOTE_C3 };

int notes2[] = {NOTE_C3, NOTE_C3, NOTE_G3, NOTE_G3, NOTE_A4, NOTE_A4,
                NOTE_G3, NOTE_F3, NOTE_F3, NOTE_E3, NOTE_E3, NOTE_D3,
                NOTE_D3, NOTE_C3 };

const byte scoreLen = sizeof(notes1)/sizeof(notes1[0]); // Количество нот

// Тоны можно объявить в виде массива
Tone notePlayer[2];

void setup(void)
{
   notePlayer[0].begin(11);
   notePlayer[1].begin(12);
}

void loop(void)
{
   for (int i = 0; i < scoreLen; i++)
   {
      notePlayer[0].play(notes1[i]);
      delay(100); // Небольшая пауза перед проигрыванием следующей ноты
      notePlayer[1].play(notes2[i]);
      delay(400);
      notePlayer[0].stop();
      notePlayer[1].stop();
      delay(30);
   }
   delay(1000);
}
```

Обсуждение работы решения и возможных проблем

Чтобы смешать два выходных сигнала разной частоты на одном динамике, сигналы с выходных контактов платы Arduino подаются на один из контактов динамика

через резисторы номиналом 500 Ом. Второй контакт динамика подключается на «землю» (контакт GND платы Arduino), как показано на схемах для предыдущих решений (например, на рис. 9.3).

На стандартных платах Arduino для генерирования первого тона задействуется таймер 2 (в результате чего подача ШИМ-сигнала на контакты 9 и 10 платы будет невозможна), а для генерирования второго сигнала — таймер 1 (что делает невозможным использование библиотеки Servo, а также подачу ШИМ-сигнала на контакты 11 и 12). На платах Mega для каждого тона используется отдельный таймер в следующем порядке: 2, 3, 4, 5, 1, 0. На момент подготовки книги эта библиотека поддерживает возможности, реализуемые архитектурой AVR, и не поддерживается платами с микроконтроллерами ARM и платами MegaAVR — такими как платы Arduino Uno WiFi R2 и Nano Every.

При проигрывании двух нот одной частоты или одной и той же ноты, но двух разных октав, может наблюдаться эффект *биения* (или *пульсации*), похожий на эффект тремоло (амплитудного вибрато). Причиной биения является не совсем идеальная синхронизация двух каналов. Этот эффект используют при ручной настройке струн гитары — биение прекращается, когда струна настроена на эталонную ноту.

Хотя на плате Arduino возможно одновременно воспроизводить три тона, а на плате Mega — более чем шесть, это создает проблемы для нормальной работы функций `delay()` и `millis()`. Поэтому для надежности рекомендуется одновременно воспроизводить только два тона на плате Arduino или пять на плате Mega.

9.4. Генерирование звуков, не лишая себя возможности использовать ШИМ-сигнал

ЗАДАЧА

Требуется генерировать звуки, но при этом не лишать себя возможности использовать ШИМ-сигнал на контактах 3 и 11.

РЕШЕНИЕ

С функцией `tone()`, к которой мы обращались в предыдущих решениях, легко работать, но для нее требуется использовать аппаратный таймер, который может быть нужен для выполнения других задач — например, для функции `analogWrite()`. Поэтому поставленная задача решается программно с помощью функции `playTone()` — вместо использования аппаратных таймеров. Применение этой функции позволяет не задействовать аппаратный таймер, но в течение проигрывания ноты код не сможет выполнять никаких других задач. В отличие от функции `tone()` языка Arduino, функция `playTone()` блокирующая — она не возвращает управление вызвавшему ее коду до тех пор, пока не будет завершено проигрывание ноты.

В листинге 9.5 приводится скетч с использованием этой функции, который проигрывает шесть нот, где частота каждой следующей ноты вдвое больше, чем преды-

дущей (на октаву выше). Ноты воспроизводятся на динамике или пьезоэлектрическом громкоговорителе, одновременно с ними изменяется яркость светодиода. Подключение динамика и светодиода к плате Arduino показано на рис. 9.4.

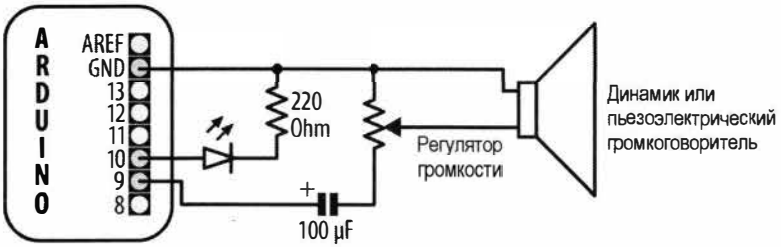


Рис. 9.4. Подключение к плате Arduino светодиода и динамика

Листинг 9.5. Одновременное воспроизведение нескольких нот с помощью функции `playTone()`

```
/*
* Скетч Tone and fade
* Проигрывает тоны, одновременно управляя яркостью светодиода
*/

byte speakerPin = 9;
byte ledPin = 3;

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

void playTone(int period, int duration)
{
   // Значение period равно одному циклу тона
   // Значение duration представляет длительность последовательности
   // импульсов в миллисекундах
   int pulse = period / 2;
   for (long i = 0; i < duration * 1000L; i += period )
   {
      digitalWrite(speakerPin, HIGH);
      delayMicroseconds(pulse);
      digitalWrite(speakerPin, LOW);
      delayMicroseconds(pulse);
   }
}

void fadeLED()
{
```

```
    // Значения этим двум статическим переменным присваивается только
    // при первом вызове этой функции
    static int brightness = 0;
    static int changeval = 5;
    analogWrite(ledPin, brightness);
    // Если превышены пределы диапазона значений функции analogWrite(),
    brightness += changeval;
    if (brightness >= 255 || brightness <= 0)
    changeval *= -1; // меняем направление изменения яркости
    delay(2);
}

void loop()
{
    // Нота с периодом в 15289 — это низкая нота С (вторая самая низкая
    // нота С на пианино)
    // Проигрываем 6 октав
    for(int period=15289; period >= 477; period=period / 2)
    {
        playTone(period, 200); // Воспроизводим тон в течение 200 мс
        fadeLED();
    }
}
```

Обсуждение работы решения и возможных проблем

Функции `playTone()` передаются два параметра: `period` и `duration` (период и длительность воспроизводимой ноты соотвстственно). Значение переменной `period` представляет время воспроизведения одного цикла тона. Один импульс подаваемого на динамик сигнала удерживается на высоком уровне, а затем на низком в течение количества микросекунд, указанного в этой переменной, в соотношении один к одному. А в цикле `for` эти импульсы повторяются в течение количества миллисекунд, заданного в переменной `duration`.

Для тех, кто предпочитает работать с частотой (`frequency`), а не с периодом сигнала, можно воспользоваться взаимосвязанностью этих характеристик: период — это просто величина, обратная частоте, т. е. единица, деленная на частоту. Значение периода должно быть в микросекундах, а т. к. одна секунда содержит 1 миллион микросекунд, то период вычисляется, как `1000000L / frequency` (буква L в конце числа указывает компилятору, что в этом вычислении нужно использовать целые числа типа `long`, чтобы не допустить превышения результатом диапазона значений целых чисел типа `int`. Целые числа типа `long` подробно рассматриваются в *разд. 2.2*):

```
void playFrequency(int frequency, int duration)
{
    int period = 1000000L / frequency;
    int pulse = period / 2;
```

Остальной код такой же, что и для функции `playTone()`:

```
    for (long i = 0; i < duration * 1000L; i += period )
    {
       digitalWrite(speakerPin, HIGH);
       delayMicroseconds(pulse);
       digitalWrite(speakerPin, LOW);
       delayMicroseconds(pulse);
    }
}
```

Код в этом решении останавливается и ожидает завершения воспроизведения тона, прежде чем выполнять какие-либо другие вычисления. Этот недостаток можно устранить, используя обработчик прерывания для исполнения кода, генерирующего тон. Исходный код функции `tone()`, входящей в состав среды Arduino IDE, содержит инструкции, как это можно сделать.

Дополнительная информация

В *разд. 9.6* рассматривается использование рассмотренных методов в более сложном проекте синтезатора звука.

Вот пара примеров синтезирования более сложных звуков с помощью платформы Arduino:

- **Кодово-импульсная модуляция**.

  Кодово-импульсная модуляция позволяет аппроксимировать аналоговые звуковые сигналы, используя цифровые сигналы. На веб-сайте Arduino Playground приводятся инструкции и код для создания 8-разрядного сигнала кодово-импульсной модуляции (**https://oreil.ly/-ddze**);

- **Шилд Pocket Piano.**

  Компания Modern Device предлагает 64-голосовой полифонический шилд синтезатора Fluxamasynth Shield для Arduino (**https://oreil.ly/qvo9h**).

9.5. Управление устройствами MIDI

ЗАДАЧА

Воспроизводить с помощью Arduino музыку на синтезаторе MIDI.

РЕШЕНИЕ

Устройство MIDI подключается к плате Arduino с помощью 5-контактного штыревого или гнездового DIN-разъема. В случае использования гнездового разъема для подключения устройства потребуется соответствующий кабель. Разъем MIDI подключается к плате Arduino, как показано в схеме на рис. 9.5. Обратите внимание на присутствие резистора номиналом 220 Ом в линии питания 5 В.

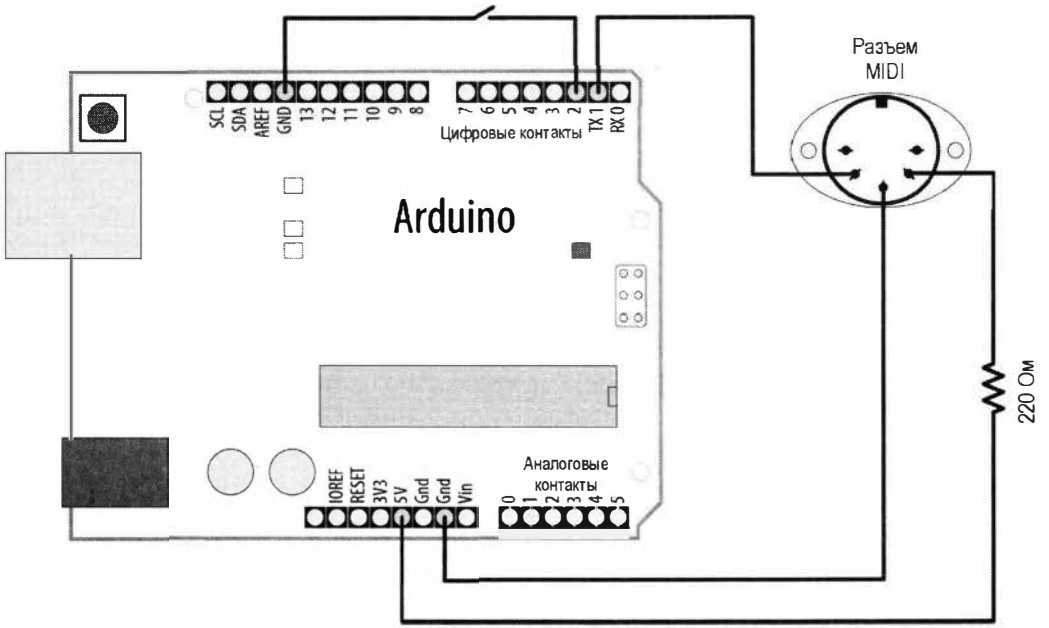


Рис. 9.5. Подключение MIDI-разъема к плате Arduino

Скетч для работы с этой схемой приводится в листинге 9.6. Код скетча следует загружать в плату Arduino при отключенном устройстве MIDI, поскольку оно может мешать загрузке. Загрузив скетч, подключите устройство MIDI к плате Arduino. Теперь при нажатии кнопки, подключенной к контакту 2, будет воспроизводиться музыкальная гамма.

Листинг 9.6. Генерирование сигналов устройства MIDI с помощью Arduino

```
/*
* Скетч midiOut
* Отправляет сообщения MIDI для проигрывания музыкальной гаммы на синтезаторе MIDI
  при каждом нажатии кнопки, подключенной к контакту 2 платы Arduino
*/

// Следующие цифры обозначают ноты для проигрывания
const byte notes[8] = {60, 62, 64, 65, 67, 69, 71, 72};
const int num_notes = sizeof(notes)/ sizeof(notes[0]);

const int switchPin = 2;
const int ledPin = LED_BUILTIN;

void setup()
{
   Serial.begin(31250);
   pinMode(switchPin, INPUT_PULLUP);
```

```
   pinMode(ledPin, OUTPUT);
}

void loop()
{
   if (digitalRead(switchPin) == LOW)
   {
      for (byte noteNumber = 0; noteNumber < num_notes; noteNumber++)
      {
         // Начинаем воспроизведение ноты
         playMidiNote(1, notes[noteNumber], 127);
         digitalWrite(ledPin, HIGH);
         delay(70); // Удерживаем ноту
         // Прекращаем воспроизведение ноты(скорость 0)
         playMidiNote(1, notes[noteNumber], 0);
         digitalWrite(ledPin, HIGH);
         delay(30);
      }
   }
}

void playMidiNote(byte channel, byte note, byte velocity)
{
   byte midiMessage= 0x90 + (channel - 1);
   Serial.write(midiMessage);
   Serial.write(note);
   Serial.write(velocity);
}
```

Обсуждение работы решения и возможных проблем

Скетч передает плате Arduino сообщения MIDI, используя контакт TX (цифровой контакт 1) последовательного интерфейса платы. Устройство, подключенное к контакту 1, может мешать загрузке кода в плату Arduino. Поэтому для загрузки скетча в плату нужно отключить провод от контакта 1 и подключить его обратно после завершения загрузки.

Интерфейс MIDI первоначально предназначался для соединения электронных музыкальных инструментов, чтобы они могли управлять друг другом. Электрические подключения и посылаемые сообщения интерфейса описываются в спецификации MIDI.

В действительности, MIDI-интерфейс является последовательным интерфейсом (с нестандартной скоростью в 31 250 бод), поэтому плата Arduino может посылать и принимать сообщения MIDI, используя свой последовательный порт на контактах 0 и 1. Поскольку при исполнении скетча последовательный порт занят для передачи сообщений MIDI, его нельзя задействовать для вывода отладочных сооб-

щений в окно монитора порта. Поэтому скетч о передаче каждой следующей ноты сообщает миганием светодиода.

Каждое сообщение MIDI состоит как минимум из одного байта, содержащего информацию о действии, которое нужно выполнить. Для некоторых команд не требуется никакой дополнительной информации, но для других нужно предоставить такую информацию, чтобы команда имела смысл. Рассматриваемый скетч передает команду проигрывания ноты, для которой требуются два дополнительных элемента информации: какую ноту проигрывать и с какой громкостью. Оба эти элемента данных имеют значение в диапазоне от 0 до 127.

Скетч инициализирует последовательный порт для работы на скорости 31 250 бод, а код для отправки команд MIDI содержится в функции `playMidiNote()`, которая приводится в листинге 9.7.

Листинг 9.7. Код функции `playMidiNote()`

```
void playMidiNote(byte channel, byte note, byte velocity)
{
   byte midiMessage= 0x90 + (channel - 1);
   Serial.write(midiMessage);
   Serial.write(note);
   Serial.write(velocity);
}
```

Эта функция принимает три параметра и вычисляет первый байт, который нужно отправить, используя информацию о канале.

Информация MIDI передается по разным каналам: от 1 до 16. Каждый канал можно задать для использования каким-либо инструментом, что позволяет проигрывать музыку, создаваемую многими разными инструментами. Команда для начала воспроизведения ноты (звука) представляет собой комбинацию битов числа `0x9n`. Старшим четырем битам этого числа присвоено постоянное двоичное значение b1001 (десятичное 9), а младшим может присваиваться значение в диапазоне от b0000 до b111, представляющее канал MIDI. Таким образом, весь байт представляет каналы посредством значений от 0 до 15 для каналов от 1 до 16, поэтому сначала вычитается 1.

Затем посылаются значение ноты и ее громкость, которая на жаргоне MIDI называется *скоростью* (velocity), поскольку исходно обозначала скорость движения клавиши музыкального инструмента.

Использование последовательных методов `write()` означает, что значения отправляются в виде байтов, а не как значения ASCII. Функция `println()` не используется по той причине, что символ возврата каретки вставлял бы в сигнал дополнительные нежелательные символы.

Воспроизведение ноты прекращается отправкой похожей команды, но со значением скорости, равным 0.

Это решение будет работать с устройствами MIDI, оснащенными пятиконтактными MIDI-разъемами стандарта DIN. С устройствами, оснащенными только USB-разъемом, скетч работать не будет, поскольку в отсутствие адаптера MIDI-to-USB он не сможет разрешить плате Arduino управлять программами синтезирования MIDI-мелодий, исполняющимися на компьютере. Хотя плата Arduino и оснащена USB-разъемом, компьютер определяет его как последовательное устройство, а не как MIDI-устройство.

Дополнительная информация

Для обмена сообщениями MIDI можно воспользоваться библиотекой MIDI для Arduino (**https://oreil.ly/zoZA_**).

Сообщения MIDI подробно рассматриваются на этой веб-странице Ассоциации MIDI: **https://oreil.ly/Oh857**.

Компания SparkFun предлагает набор для сборки шилда MIDI (**https://oreil.ly/faK0c**), содержащий входные и выходные разъемы MIDI, несколько кнопок и оптоизолятор для обеспечения электрической развязки между устройством MIDI и платой Arduino.

Плату Teensy можно запрограммировать для функционирования как устройство MIDI с собственным USB-интерфейсом.

9.6. Создание синтезатора звуков

ЗАДАЧА

Требуется генерировать сложные звуки, подобные тем, что синтезируются при создании электронной музыки.

РЕШЕНИЕ

Эмуляция генераторов колебаний звуковой частоты, используемая в синтезаторе звуков, — весьма непростой процесс, но скетч Auduino разработки Питера Найта (Peter Knight) позволяет Arduino синтезировать более сложные и интересные звуки. Поскольку в скетче используется много низкоуровневых возможностей, он вряд ли сможет исполняться на каких-либо иных платах, кроме 8-разрядных плат с микроконтроллером ATmega — например, на плате Arduino Uno.

Скетч можно загрузить по этой ссылке: **https://oreil.ly/JwHYy**.

Для работы скетча к контактам с 0 по 4 платы Arduino нужно подключить пять линейных потенциометров номиналом 4,7 кОм, как показано на рис. 9.6. Лучше взять потенциометры с длинными валами, а не с короткими, поскольку их легче вращать пальцами для тонкой корректировки настроек. Создаваемый скетчем звуковой сигнал снимается с контакта 3 и подается на усилитель низкой частоты.

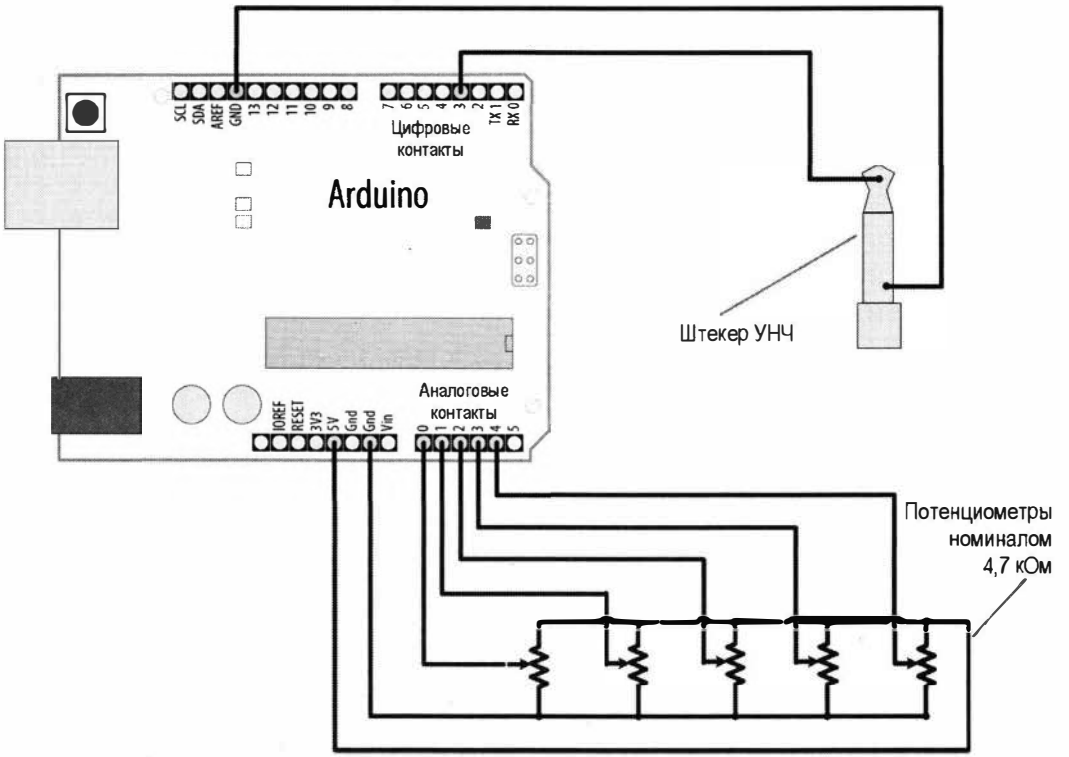


Рис. 9.6. Схема подключения потенциометров и выходного сигнала для скетча синтезатора звуков

Напряжение величиной 5 В превышает уровень напряжения входного сигнала, ожидаемый усилителем низкой частоты, поэтому может потребоваться уменьшить его с помощью потенциометра номиналом 4,7 кОм. Один вывод этого потенциометра нужно подключить к контакту 9 платы Arduino, а другой — к контакту «земли» (Gnd). Выходной сигнал со среднего вывода потенциометра подается на вход усилителя — подключите этот вывод к центральному контакту штекера. Наружный контакт штекера подключается к «земле».

Обсуждение работы решения и возможных проблем

Код скетча сложный, поскольку он непосредственно манипулирует аппаратными таймерами, чтобы генерировать требуемые частоты, которые затем обрабатываются программно для создания различных звуковых эффектов. Листинг кода здесь не приводится, поскольку для того, чтобы использовать его с Arduino, разбираться в нем нет надобности.

Для генерирования звука платой Arduino применяется метод *гранулярного синтеза*, заключающийся в использовании создаваемых электронным способом источников звука, называющихся *гранулами*. Частота и скорость затухания каждой гранулы (вводы 0 и 2 для одной гранулы и 3 и 1 для другой) управляются с помощью потенциометров. Синхронизация между гранулами управляется вводом 4.

При желании можно попробовать поэкспериментировать с кодом, изменив гамму для вычисления частоты. По умолчанию используется пентатонная гамма, но ее можно закомментировать и раскомментировать строку кода для использования другой гаммы.

При добавлении кода в главный цикл `loop()` следует быть осторожным, поскольку скетч высоко оптимизирован и дополнительный код может слишком замедлить его исполнение, что отрицательно скажется на процессе синтезирования звука.

Любой из потенциометров можно заменить датчиком, создающим аналоговый выходной сигнал (см. *главу 6*). Например, фоторезистор (см. *разд. 6.3*) или датчик расстояния с аналоговым выходным сигналом (рассмотренный в конце *разд. 6.5*), подключенный к одному из аналоговых контактов (A0 или A3), позволит управлять тембром звука, приближая или удаляя руку от датчика (почитайте в Википедии или в другом источнике, который можно найти, выполнив соответствующий поиск в Google, о музыкальном инструменте терменвокс, на котором играют, перемещая над ним руки).

Дополнительная информация

Видеодемонстрацию действия Auduino можно посмотреть на этой веб-странице: **https://oreil.ly/hHL5H**.

Гранулярный синтез объясняется в этих статьях Википедии: **https://oreil.ly/kPaLq** (англоязычная) и **https://ru.wikipedia.org/wiki/Гранулярный_синтез** (русскоязычная).

Электронный музыкальный инструмент терменвокс описан в этих статьях Википедии: **https://oreil.ly/LjVSt** (англоязычная) и **https://ru.wikipedia.org/wiki/Терменвокс** (русскоязычная).

9.7. Синтез звуков высокого качества

ЗАДАЧА

Требуется генерировать звуки более высокого качества, чем те, которые удавалось получать с помощью прямоугольных сигналов, создаваемых с помощью библиотеки Tone на 8-разрядной плате без дополнительного оборудования. Например, вы хотите воспроизводить музыкальные мелодии на основе синусоидальных сигналов, проигрывать файлы WAV и синтезировать высококачественные звуки, не прибегая к использованию дополнительного оборудования.

РЕШЕНИЕ

Как отмечалось в *разд. 1.8*, платы с микроконтроллером SAMD оснащены цифроаналоговым преобразователем (ЦАП), который может генерировать настоящий аналоговый сигнал с напряжением в диапазоне от 0 В и до величины рабочего напряжения (3,3 В) платы. Использование возможностей ЦАП, вкупе с более высокой

скоростью работы 32-разрядной платы, позволяет создавать сложные звуковые сигналы. Подключите к плате пьезоэлектрический громкоговоритель, как показано ранее на рис. 9.2, но только не к контакту 9, а контакту вывода сигнала ЦАП — это контакт A0 для плат Arduino Zero, Adafruit Metro M0 и RedBoard Turbo компании SparkFun. Установите библиотеку синтеза звуков Mozzi (в *разд 16.2* показано, как подключаются сторонние библиотеки) и загрузите в плату скетч из листинга 9.8.

Листинг 9.8. Синтезирование звуков высокого качества

```
/*
* Скетч Mozzi Melody
* Проигрывает мелодию часов Большой Бен
*/

#include <MozziGuts.h>
#include <Oscil.h> // Шаблон генератора колебаний
#include <tables/sin2048_int8.h> // Таблица синусов для генератора колебаний
#include <EventDelay.h>
#include <mozzi_midi.h>

#define CONTROL_RATE 64 // Скорость управления в Гц – используем степени 2

enum notes
{
   E3 = 52, B4 = 59, E4 = 64, F4S = 66, G4S = 68, REST = 0
};

int score[] = { E4, G4S, F4S, B4, REST,
                E4, F4S, G4S, E4, REST,
                G4S, E4, F4S, B4, REST,
                B4, F4S, G4S, E4, REST,
                E3, REST, E3, REST, E3, REST, E3, REST };
const byte scoreLen = sizeof(score) / sizeof(score[0]);

byte beats[scoreLen] = {2, 2, 2, 2, 2,
                        2, 2, 2, 2, 2,
                        2, 2, 2, 2, 2,
                        2, 2, 2, 2, 10,
                        4, 4, 4, 4, 4, 4, 4, 10};
unsigned int beat_ms = 60000 / 180; // Время 1/8 ноты в миллисекундах

const int pauseTime = 200; // Пауза между нотами
int currNote = 0;          // Указатель воспроизводимой ноты
bool pausing = false;      // Кратковременная пауза между тактами
// Синусоидальный сигнал
Oscil <SIN2048_NUM_CELLS, AUDIO_RATE> aSin(SIN2048_DATA);
EventDelay kChangeNoteDelay; // Объект задержки
```

```
void setup()
{
   startMozzi(CONTROL_RATE);
   kChangeNoteDelay.start();
}

void updateControl()
{
   if (kChangeNoteDelay.ready())
   {
      // Время задержки истекло?
      pausing = !pausing; // Переключаем состояние покоя на обратное
      if (pausing)
      {
         aSin.setFreq(0); // Устанавливаем частоту
         kChangeNoteDelay.set(pauseTime); // Выдерживаем паузу в 200 мс
      }
      else
      {
         if (currNote >= scoreLen)
         {
            currNote = 0; // После завершения переходим в начало
         }
         int duration = beats[currNote] * beat_ms;    // Получаем
                                                      // длительность из массива
         kChangeNoteDelay.set(duration - pauseTime); // Задаем
                                                      // длительность звучания ноты
         aSin.setFreq(mtof(score[currNote])); // Задаем частоту ноты
         currNote++;
      }
      kChangeNoteDelay.start();
   }
}

int updateAudio()
{
   return aSin.next();
}

void loop()
{
   audioHook(); // Код для главного цикла loop()
}
```

Обсуждение работы решения и возможных проблем

Библиотека Mozzi предназначена для синтеза сложных звуков на платах Arduino и совместимых платах. Хотя библиотека поддерживает 8-разрядные платы, на них

она использует ШИМ-сигналы, что ограничивает качество звука. Но на 32-разрядных платах — например, платах с микроконтроллером SAMS, таких как плата Teensy, — эта библиотека способна воспроизводить звуки высокого качества. Библиотека Mozzi представляет собой мощную платформу, поддерживающую многие возможности и содержащую десятки примеров программ.

В начале скетча выполняются организационные задачи, состоящие из подключения заголовочных файлов и определения скорости управления библиотеки, которая определяет частоту вызовов функций обновления. Затем выполняется определение перечисления (`enum`) нот, используемых в скетче. Для удобства используются ноты MIDI (см. *разд. 9.5*). Поскольку ноты MIDI можно представлять в виде целых чисел, это позволяет использовать их в перечислении, что, в свою очередь, упрощает использование их символических идентификаторов в массиве целых чисел. То же самое можно было бы сделать, используя операторы `#define`, — в этом случае вы могли бы указывать частоты числами с плавающей запятой вместо использования нот MIDI.

Подобно скетчу из *разд. 9.2*, в этом скетче ноты мелодии сохраняются в массиве, также в массиве сохраняется и информация о длительности в тактах каждой ноты. Затем выполняется инициализация некоторых переменных и объектов, которые используются далее в скетче.

В функции `setup()` выполняется инициализации системы Mozzi и запускается таймер. С этого места все начинает заметно отличаться от типичного скетча Arduino. Вместо использования функции `delay()`, определяющей время ожидания для воспроизведения следующей ноты, все изменения происходят в функции `updateControl()`. По истечении времени таймера скетч меняет состояние ожидания на обратное, используемое для определения момента начала выдерживания кратковременной паузы между нотами. Если значение булевой переменной `pausing` равно `true`, выдерживается пауза в 200 мс. В противном случае выполняется переход к следующей ноте в массиве `score`, вычисляется длительность ее звучания, а также задается ее частота путем преобразования номера MIDI ноты в ее частоту в герцах. Затем устанавливается время таймера на длительность звучания ноты (вычитая время паузы, чтобы придерживаться стандартного музыкального тактирования).

Дополнительная информация

Совместимый с Arduino набор музыкального синтезатора, состоящий из платы ArduTouch и программного обеспечения, предоставляет возможности, подобные возможностям библиотеки Mozzi. Программное обеспечение ArduTouch можно исполнять и на плате Arduino Uno.

Библиотека Teensy Audio (**https://oreil.ly/g1p93**) обеспечивает поддержку плат Teensy для работы со звуками высокого качества. Библиотека поддерживает полифонические возможности, а также возможности записи, синтеза, фильтрации звука и много чего другого.

ГЛАВА 10

Дистанционное управление внешними устройствами

10.0. Введение

Плата Arduino может взаимодействовать практически с любым устройством, оснащенным контроллером дистанционного управления какого-либо типа, включая телевизоры, аудиооборудование, фотокамеры, гаражные двери, бытовые устройства и игрушки. Работа большинства систем дистанционного управления основана на пересылке цифровых данных от передатчика приемнику с помощью инфракрасного (ИК) света или беспроводной радиосвязи. Для преобразования нажатия кнопок на пульте в цифровой сигнал используются различные протоколы (шаблоны сигналов), и в решениях этой главы рассматривается применение наиболее распространенных систем и протоколов дистанционного управления применительно к возможностям платы Arduino.

Пульт дистанционного управления (ДУ) посылает удаленному устройству команды, включая и выключая инфракрасный светодиод, создавая тем самым уникальные последовательности включенных и выключенных состояний (коды), содержащие обычно от 12 до 32 битов (элементов информации). Каждой кнопке на пульте дистанционного управления сопоставлен определенный код, который передается при нажатии этой кнопки. При удерживании кнопки нажатой ее код обычно передается непрерывно раз за разом, хотя некоторые пульты ДУ (например, компании NEC) в таком случае передают специальный код повтора. А на пультах ДУ RC-5 и RC-6 компании Philips при каждом нажатии кнопки значение одного бита кода меняется на обратное. По состоянию этого переключаемого бита приемник определяет повторное нажатие кнопки. Дополнительная информация о технологиях, используемых в инфракрасных системах дистанционного управления, приводится на веб-сайте SB-Projects (**https://oreil.ly/BeSzk**).

В решениях инфракрасного дистанционного управления этой главы используется недорогой модуль ИК-приемника, принимающий коды, передаваемые пультом ДУ, и преобразующий их в цифровые сигналы, которые можно подавать на плату Arduino для дальнейшей обработки. Эта обработка заключается в декодировании сигнала с помощью разработанной Кеном Ширriффом (Ken Shirriff) библиотеки IRremote, которая устанавливается стандартным способом с помощью Менеджера библиотек среды Arduino IDE (подробно об установке библиотек сторонних разработчиков рассказано в *разд. 16.2*).

Та же библиотека используется и в скетчах решений, где Arduino отправляет команды, действуя в качестве пульта дистанционного управления.

Эмулирование систем ДУ, использующих беспроводную радиосвязь, сопряжено с бóльшими трудностями, чем эмулирование аналогичных ИК-систем. Однако контакты кнопок пультов ДУ этого типа можно активировать с помощью Arduino. В решениях для дистанционного управления на основе беспроводной радиосвязи нажатие кнопки на пульте имитируется замыканием контактов внутри пульта. Для этого может потребоваться разобрать пульт ДУ и подключить выводы контактов его кнопок к контактам платы Arduino. Электрическая развязка между пультом ДУ и платой Arduino обеспечивается специальными электронными устройствами — *оптоизоляторами* (оптронами), вследствие чего напряжения платы Arduino не могут повредить пульт ДУ и наоборот.

Оптоизоляторы позволяют безопасно управлять схемой, уровни рабочих напряжений которой могут отличаться от напряжений платы Arduino. Как можно судить по части «изолятор», оптоизоляторы обеспечивают электрическую изоляцию одного устройства от другого. Это достигается заменой одного участка электрической цепи оптическим звеном, состоящим из светодиода и фототранзистора. Электрический сигнал платы Arduino управляет светодиодом, который преобразовывает его в оптический сигнал. Этот оптический сигнал принимается фототранзистором, который преобразовывает его обратно в электрический сигнал.

Таким образом обеспечивается прохождение электрического сигнала от одного устройства к другому, но при этом между этими устройствами отсутствует электрическая связь. Нажатие кнопки в схеме Arduino включает светодиод, что, в свою очередь, активирует фототранзистор, который замыкает цепь между контактами кнопки пульта ДУ, что эквивалентно нажатию этой кнопки.

10.1. Реагирование на команды ИК-пульта ДУ

ЗАДАЧА

Требуется реагировать на нажатие кнопки пульта ДУ телевизором или другим устройством.

РЕШЕНИЕ

Для приема платой Arduino сигналов пульта ДУ используется модуль приемника ИК-сигналов. На рынке широко представлены приемники ИК-сигнала TSOP38238, TSOP4838, PNA4602 и TSOP2438. Первые три эти приемника имеют одинаковую схему расположения контактов, а у приемника TSOP2438 выводы питания (+5 В и GND) поменяны местами. При использовании какого-либо иного приемника ИК-сигналов уточните схему расположения его контактов в справочном листке (datasheet) на него и подключите его к плате Arduino, как там указано. Подключение приемников TSOP38238, TSOP4838 и PNA4602 к плате Arduino показано в схеме на рис. 10.1.

Скетч для работы с этой схемой приводится в *листинге 10.1*. Скетч переключает состояние встроенного светодиода при нажатии любой кнопки на инфракрасном

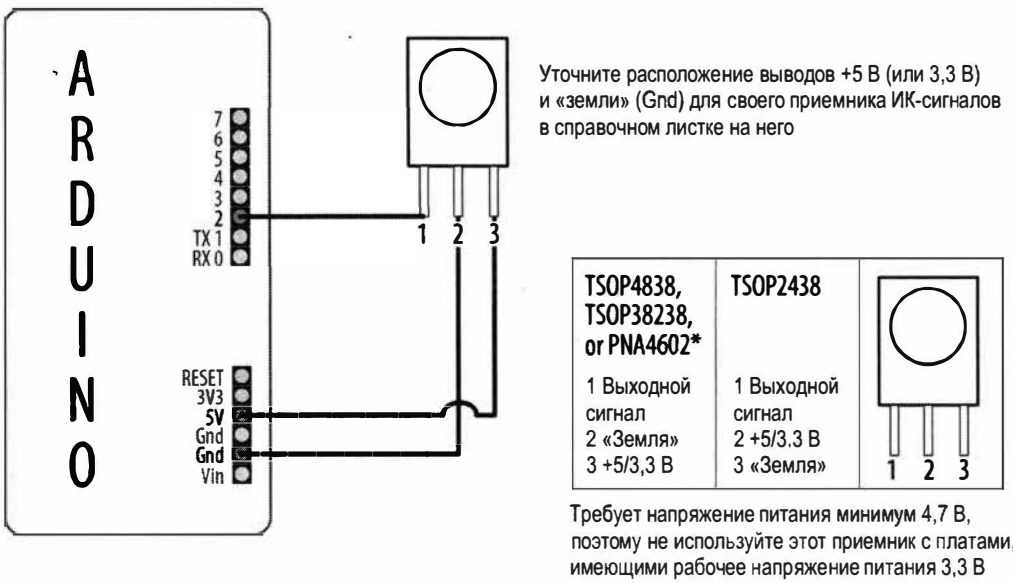


Рис. 10.1. Подключение приемника ИК-сигнала к плате Arduino

пульте ДУ. В скетче используется библиотека IRremote, которая устанавливается стандартным способом с помощью менеджера библиотек.

Листинг 10.1. Скетч для реагирования на нажатие кнопки ИК-пульта ДУ

```
/*
Скетч IR_remote_detector
Приемник ИК-сигнала подключен к контакту 2 платы Arduino
Встроенный светодиод платы переключает состояние при каждом нажатии
любой кнопки на пульте ДУ
*/

#include <IRremote.h> // Подключаем код библиотеки к скетчу

const int irReceiverPin = 2; // Приемник ИК-сигнала подключен
                             // к контакту 2 платы Arduino
const int ledPin = LED_BUILTIN;

IRrecv irrecv(irReceiverPin); // Создаем экземпляр объекта IRrecv
decode_results decodedSignal; // Переменная для хранения результата,
                              // полученного с ИК-приемника

void setup()
{
   pinMode(ledPin, OUTPUT);
   irrecv.enableIRIn(); // Инициализируем объект ИК-приемника
}
```

```
bool lightState = LOW; // Отслеживаем, включен ли светодиод
unsigned long last = millis(); // Запоминаем, когда было получено
                               // последнее ИК-сообщение

void loop()
{
   if (irrecv.decode(&decodedSignal) == true) // true, если было получено сообщение
   {
      if (millis() - last > 250)
      {
         // Прошла ли 1/4 секунды после приема последнего сообщения?
         if (lightState == LOW)
            lightState = HIGH;
         else
            lightState = LOW;
         lightState = lightState ; // Да: переключаем состояние светодиода
         digitalWrite(ledPin, lightState);
      }
      last = millis();
      irrecv.resume(); // Ожидаем другое сообщение
   }
}
```

При использовании платы, рассчитанной на напряжение 3,3 В, которая не может работать с логическими уровнями величиной 5 В, на ИК-приемник следует подавать напряжение питания 3,3 В, а не 5 В.

Обсуждение работы решения и возможных проблем

Приемник ИК-сигналов преобразовывает полученные сигналы в последовательность цифровых импульсов: высоких и низких уровней (единиц и нулей), соответствующих коду кнопки пульта ДУ. Библиотека IRremote декодирует эти последовательности в числовое значение нажатой кнопки пульта ДУ (получаемые скетчем значения будут зависеть от конкретного используемого пульта).

Строка кода:

```
#include <IRremote.h>
```

в начале скетча подключает к нему код указанной библиотеки, а строка кода:

```
IRrecv irrecv(irReceiverPin);
```

создает экземпляр класса `IRrecv` с названием `irrecv` для приема сигналов от ИК-приемника, подключенного к контакту `irReceiverPin` (контакт 2 в нашем случае). Более подробная информация по использованию библиотек излагается в *главе 16*.

Объект `irrecv` используется для доступа к сигналу, поступающему с ИК-приемника. Объекту можно задавать команды для получения и декодирования ИК-сигналов. Результаты декодирования, предоставляемые библиотекой, сохраняются в пере-

менной decode_results. Объект ИК-приемника (irrecv) инициализируется в функции setup() строкой кода:

```
irrecv.enableIRIn();
```

Результаты декодирования проверяются в главном цикле loop(), вызывая метод (функцию):

```
irrecv.decode(&decodedSignal)
```

При наличии данных функция decode() возвращает булево значение true, которое сохраняется в переменной decodedSignal. В *разд. 2.11* рассматривается использование символа амперсанда в вызовах функций, чтобы модифицированные в них значения параметров можно было передать обратно в вызывающий код.

При получении сообщения от пульта ДУ код меняет состояние светодиода на обратное, если прошло более четверти секунды со времени последнего такого изменения состояния. Эта задержка необходима, чтобы избежать многократного включения и выключения светодиода в случае использования пультов ДУ, отправляющих многократные сообщении при одном нажатии кнопки, что может создать впечатление, что светодиод включается и выключается произвольно.

Переменная decodedSignal будет содержать значение, сопоставленное нажатой кнопке пульта ДУ. В рассматриваемом решении это значение игнорируется, хотя оно используется в следующем решении. При желании это значение можно выводить в окно монитора порта, добавив в скетч решения оператор Serial.println(), выделенный полужирным шрифтом в следующем фрагменте кода (при этом в функцию setup() также нужно будет добавить оператор Serial.begin(9600);).

```
if (irrecv.decode(&decodedSignal) == true) // true, если было получено сообщение
{
   if (millis() - last > 250)
   {
      // Прошла ли 1/4 секунды после приема последнего сообщения?
      Serial.println(decodedSignal.value);
```

Объекту ИК-приемника нужно дать указание продолжать ожидание поступления сигналов, что осуществляется оператором:

```
irrecv.resume();
```

В рассматриваемом случае скетч реагирует на нажатие кнопки пульта ДУ включением или выключением светодиода, но вместо этого он может управлять другими устройствами — например, шаговым электродвигателем, который вращает ручку управления яркостью светильника или громкости стереосистемы (более подробно об управлении внешними устройствами рассказано в *главе 8*).

Дополнительная информация

Подробную информацию по инфракрасным сигналам управления вы найдете в разделе **Infrared** блога Кена Ширриффа (Ken Shirriff) по адресу: **https://oreil.ly/Jkryi**.

10.2. Декодирование ИК-сигналов управления пульта ДУ

ЗАДАЧА

Требуется определить нажатие определенной кнопки пульта ДУ телевизором или другим устройством.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 10.2. Скетч реагирует на нажатие кнопки пульта ДУ изменением яркости светодиода. В начале исполнения скетча запрашивается нажатие кнопок пульта ДУ от нулевой (0) до 4-й. Коды этих кнопок сохраняются в памяти RAM платы Arduino. Скетч реагирует на нажатие кнопки, изменяя яркость светодиода в соответствии с нажатой кнопкой: кнопка 0 выключает светодиод, а кнопки с 1-й по 4-ю задают соответственно увеличивающуюся яркость.

Листинг 10.2. Управление яркостью светодиода посредством ИК-пульта ДУ

```
/*
* Скетч RemoteDecode
* Декодирует ИК-сигналы пульта ДУ для управления яркостью светодиода
* Определяет и сохраняет коды кнопок с 0-й по 4-ю в начале работы
* Кнопка 0 выключает светодиод. Кнопки с 1-й по 4-ю устанавливают
  соответствующую яркость светодиода
*/

#include <IRremote.h> // Подключаем библиотеку IRremote

const int irReceivePin = 2; // Номер контакта, на который подается
                            // входной сигнал с ИК-приемника
const int ledPin = 9;       // Номер ШИМ-контакта, к которому подключен светодиод
const int numberOfKeys = 5; // Запоминаем коды 5 кнопок (с 0-й по 4-ю)

long irKeyCodes[numberOfKeys]; // Массив для хранения кодов кнопок

IRrecv irrecv(irReceivePin); // Создаем экземпляр объекта ИК-приемника
decode_results results;      // Полученные данные ИК-сигнала сохраняются
                             // в этой переменной

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Для плат Leonardo и ARM
   pinMode(irReceivePin, INPUT);
   pinMode(ledPin, OUTPUT);
```

```
   irrecv.enableIRIn(); // Инициализируем объект ИК-приемника
   learnKeycodes();     // Запоминаем коды кнопок пульта ДУ
   Serial.println("Press a remote key"); // Нажмите кнопку на пульте ДУ
}

void loop()
{
   long key;
   int brightness;

   if (irrecv.decode(&results))
   {
      // Переходим сюда при получении данных
      irrecv.resume();
      key = convertCodeToKey(results.value);
      if(key >= 0)
      {
         Serial.print("Got key "); // Получен код кнопкки
         Serial.println(key);
         brightness = map(key, 0,numberOfKeys-1, 0, 255);
         analogWrite(ledPin, brightness);
      }
   }
}

/*
* Запоминаем коды управления кнопок пульта ДУ
*/
void learnKeycodes()
{
   while(irrecv.decode(&results)) // Очищаем буфер
   irrecv.resume();
   Serial.println("Ready to learn remote codes");
                  // Все готово для запоминания кодов кнопок пульта ДУ
   long prevValue = -1;
   int i=0;
   while(i < numberOfKeys)
   {
      Serial.print("press remote key "); // Нажмите кнопку на пульте ДУ
      Serial.print(i);
      while(true)
      {
         if(irrecv.decode(&results))
         {
            if(results.value != -1 &&
            results.decode_type != UNKNOWN &&
            results.value != prevValue)
```

```
            {
               showReceivedData();
               Serial.println(results.value);
               irKeyCodes[i] = results.value;
               i = i + 1;
               prevValue = results.value;
               irrecv.resume(); // Получаем код следующей кнопки
               break;
            }
            irrecv.resume(); // Получаем код следующей кнопки
         }
      }
   }
   Serial.println("Learning complete"); // Запоминание завершено
}

/*
* Преобразовывает код кнопки пульта ДУ в номер кнопки
* (или -1, если не получено никакого кода)
*/
int convertCodeToKey(long code)
{
   for(int i=0; i < numberOfKeys; i++)
   {
      if(code == irKeyCodes[i])
      {
         return i; // Получен код кнопки, возвращаем его
      }
   }
   return -1;
}

/*
* Отображаем тип протокола и значение
*/
void showReceivedData()
{
   if (results.decode_type == UNKNOWN)
   {
      Serial.println("-Could not decode message");
                  // Не удалось декодировать команду
   }
   else
   {
      if (results.decode_type == NEC)
      {
         Serial.print("- decoded NEC: ");
      }
```

```
    else if (results.decode_type == SONY)
    {
      Serial.print("- decoded SONY: ");
    }
    else if (results.decode_type == RC5)
    {
      Serial.print("- decoded RC5: ");
    }
    else if (results.decode_type == RC6)
    {
      Serial.print("- decoded RC6: ");
    }
    Serial.print("hex value = ");
    Serial.println( results.value, HEX);
  }
}
```

Обсуждение работы решения и возможных проблем

Скетч решения основан на использовании библиотеки IRremote, более подробная информация о которой приводится в *разд. 10.0* этой главы.

В начале работы скетч инициализирует экземпляр объекта ИК-приемника следующей строкой кода:

```
irrecv.enableIRIn(); // Инициализируем объект ИК-приемника
```

Затем вызывается функция `learnKeycodes()`, которая предлагает пользователю последовательно нажать кнопки пульта ДУ с 0-й по 4-ю. Код для каждой кнопки сохраняется в массиве `irKeyCodes`. После запоминания кодов всех требуемых кнопок начинает исполняться код главного цикла `loop()`, который ожидает нажатия кнопки и проверяет, не является ли она одной из кнопок, сохраненных в массиве `irKeyCodes`. При положительном результате проверки это значение используется для установки соответствующего уровня яркости светодиода путем записи его на контакт управления светодиодом с помощью функции `analogWrite()`.

Дополнительная информация по использованию функций `map()` и `analogWrite()` для управления яркостью светодиода приводится в *разд. 5.7*.

Библиотеку IRremote можно использовать с большинством ИК-пультов дистанционного управления. Она может определять временны́е характеристики сигналов, а затем повторять запомненные сигналы.

Запомненные коды кнопок можно сохранить в постоянной памяти, чтобы не выполнять процесс запоминания при каждом запуске скетча. Для этого нужно заменить код объявления массива `irKeyCodes` кодом из листинга 10.3, который инициализирует значение каждой кнопки, а также закомментировать вызов функции `learnKeycodes()`. Замените коды кнопок кодами для используемого вами пульта ДУ.

Эти коды отображаются в окне монитора порта при нажатии кнопок в процессе исполнения функции `learnKeycodes()` в исходном скетче.

Листинг 10.3. Объявление массива для постоянного хранения кодов кнопок

```
long irKeyCodes[numberOfKeys] =
{
   0x18E758A7, //0 key
   0x18E708F7, //1 key
   0x18E78877, //2 key
   0x18E748B7, //3 key
   0x18E7C837, //4 key
};
```

Дополнительная информация

В *разд. 18.1* приводится дополнительная информация о том, как сохранять полученные данные в энергонезависимой памяти ЭСППЗУ.

10.3. Имитация ИК-сигналов управления пульта ДУ

ЗАДАЧА

Требуется управлять с помощью платы Arduino телевизором или каким-либо другим устройством, эмулируя коды ИК-сигналов пульта ДУ. Эта задача противоположна рассмотренной в *разд. 10.2* — здесь нам нужно не принимать команды, а отправлять их.

РЕШЕНИЕ

Для решения этой задачи к плате Arduino нужно подключить пять кнопок и инфракрасный светодиод, как показано на рис. 10.2.

Скетч для управления этой схемой приводится в листинге 10.4. Для управления устройством в нем используются коды кнопок из скетча, приведенного в решении из *разд. 10.2* (см. листинг 10.2). Коды для управления вашим устройством, скорее всего, будут другими, поэтому узнайте их, исполнив скетч из *разд. 10.2*, а затем замените ими коды в скетче из листинга 10.4. Нажатие одной из пяти кнопок выбирает и отправляет требуемый код.

Листинг 10.4. Скетч для эмулирования ИК-пульта ДУ

```
/*
* Скетч irSend
* Требует подключения ИК-светодиода к контакту 3
* и 5 кнопок к контактам 6-10
*/
```

```
#include <IRremote.h> // Подключаем библиотеку IRremote

const int numberOfKeys = 5;
const int firstKey = 6; // Номер первого из пяти последовательных
                        // контактов, к которым подключаются кнопки

bool buttonState[numberOfKeys];
bool lastButtonState[numberOfKeys];

long irKeyCodes[numberOfKeys] =
{
   0x18E758A7, //0 key
   0x18E708F7, //1 key
   0x18E78877, //2 key
   0x18E748B7, //3 key
   0x18E7C837, //4 key
};

IRsend irsend;

void setup()
{
   for (int i = 0; i < numberOfKeys; i++)
   {
      buttonState[i] = true;
      lastButtonState[i] = true;
      int physicalPin=i + firstKey;
      pinMode(physicalPin, INPUT_PULLUP); // Включаем внутренние повышающие резисторы
   }
   Serial.begin(9600);
}

void loop()
{
   for (int keyNumber=0; keyNumber<numberOfKeys; keyNumber++)
   {
      int physicalPinToRead = keyNumber + firstKey;
      buttonState[keyNumber] = digitalRead(physicalPinToRead);
      if (buttonState[keyNumber] != lastButtonState[keyNumber])
      {
         if (buttonState[keyNumber] == LOW)
         {
            irsend.sendSony(irKeyCodes[keyNumber], 32);
            Serial.println("Sending"); // Посылаем команду
         }
         lastButtonState[keyNumber] = buttonState[keyNumber];
      }
   }
}
```

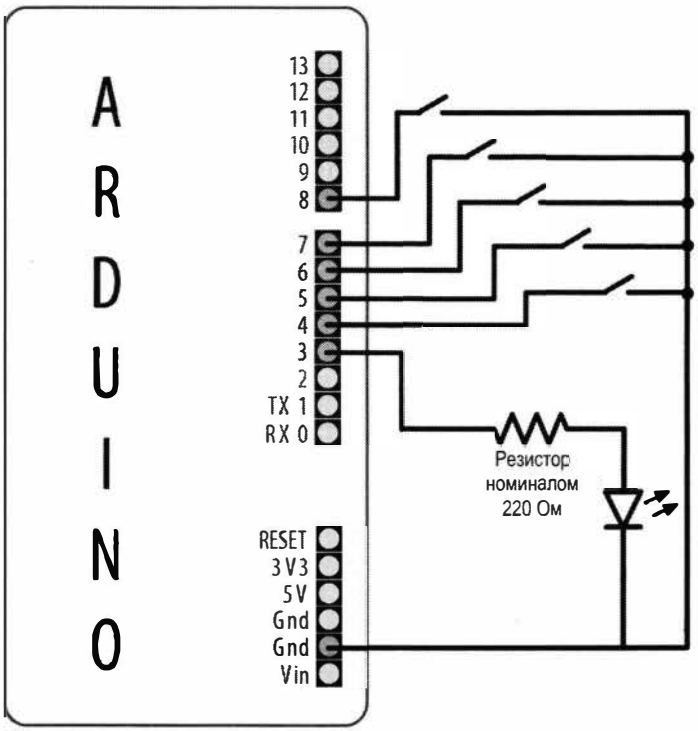


Рис. 10.2. Подключение кнопок и инфракрасного светодиода к плате Arduino

Увидеть невооруженным глазом отправляемые коды невозможно, поскольку инфракрасный свет невидим для человеческого глаза.

Но в работе передатчика можно удостовериться, наблюдая за ИК-светодиодом на жидкокристаллическом дисплее видоискателя цифровой камеры.

Обсуждение работы решения и возможных проблем

Скетч Arduino управляет удаленным устройством, включая и выключая инфракрасный светодиод таким образом, чтобы имитировать сигналы, посылаемые ИК-пультом ДУ от этого устройства. Характеристики используемого ИК-светодиода не являются критическими. Информация по подбору подходящего компонента приводится в *приложении 1*.

Преобразование числовых кодов в мигания ИК-светодиода реализует библиотека IRremote. Для этого в скетче требуется создать экземпляр объекта этой библиотеки. Такой экземпляр для управления светодиодом, подключенным к контакту 3 (выбирать контакт для подключения светодиода не представляется возможным, поскольку этот параметр жестко закодирован в библиотеке), создается следующей строкой кода:

```
IRsend irsend;
```

В зависимости от используемой платы Arduino, ИК-светодиод может потребоваться подключить к иному контакту, чем 3. Например, для плат Teensy 3.x ИК-светодиод надо подключать к контакту 5. Более подробная информация на этот счет содержится в файле *README* библиотеки IRremote (**https://oreil.ly/XOfdd**). Если для вашей платы ИК-светодиод нужно подключать к одному из контактов, который в скетче решения используется для подключения кнопок (контакты 6–10), следует взять другой диапазон номеров контактов для подключения кнопок.

Для хранения диапазона значений кодов управления в скетче используется массив `irKeyCodes` (см. также решение *разд. 2.4*). Скетч отслеживает состояние пяти кнопок и при обнаружении нажатия одной из них посылает соответствующий код кнопки следующим оператором:

```
irsend.sendSony(irKeyCodes[keyNumber], 32);
```

Объект `irSend` поддерживает разные функции для работы с различными популярными форматами ИК-кодов. В случае использования системы дистанционного управления с другими форматами кодов, посмотрите эти форматы в документации на библиотеку. Коды также можно узнать с помощью скетча из *разд. 10.2*, который выводит их в окно монитора порта.

При передаче функции кода кнопки из массива в конце кода также передается число, сообщающее функции количество битов в коде. Префикс `0x` перед определениями кодов в массиве `irKeyCodes` в начале скетча означает шестнадцатеричный формат этих чисел (подробная информация о шестнадцатеричных числах приводится в *главе 2*). Каждый символ шестнадцатеричного числа представляет 4-битное значение. Восемь шестнадцатеричных цифр кода означает 32 бита.

Светодиод к плате Arduino подключается через токоограничивающий резистор (подробная информация о применении токоограничивающих резисторов со светодиодами приводится во *введении главы 7* (*разд. 7.0*)).

Зону действия передатчика можно увеличить, используя несколько светодиодов или более мощный светодиод.

Дополнительная информация

Дополнительная информация по управлению светодиодами приведена в *главе 7*.

Проект необычного ИК-пульта дистанционного управления для телевизоров представлен на веб-странице **https://oreil.ly/7BzKh**.

10.4. Управление цифровой фотокамерой

ЗАДАЧА

Требуется с помощью платы Arduino управлять цифровой фотокамерой, чтобы можно было программным способом управлять ее работой. Например, делать регулярные периодические снимки или делать снимки по событию, выявленному датчиком, подключенным к плате Arduino.

РЕШЕНИЕ

Эту задачу можно решить несколькими способами. Если камера оснащена инфракрасной системой дистанционного управления, тогда можно воспользоваться решением из *разд. 10.2*, чтобы выяснить релевантные коды ее пульта ДУ, а затем применить решение из *разд. 10.3*, чтобы посылать эти коды на камеру.

Если же вместо ИК-системы дистанционного управления камера оснащена разъемом для подключения модуля дистанционного управления затвором, можно использовать этот разъем.

Разъем TRS[1] для управления затвором камеры обычно имеет диаметр 2,5 или 3,5 мм, но длина и форма кончика штекера могут быть нестандартными. Самый лучший способ получить требуемый штекер для гнездового разъема камеры — это приобрести недорогой проводной модуль дистанционного управления для используемой модели камеры и модифицировать его, как требуется, или же купить специальный адаптерный кабель. Такой кабель можно найти, выполнив поиск по ключевой фразе `TRS camera shutter` или `пульт дистанционного спуска затвора`.

Плата Arduino подключается к разъему спуска затвора камеры через оптоизоляторы (оптроны), как показано на рис. 10.3.

Скетч для управления камерой приводится в листинге 10.5. Скетч делает 20 снимков, по одному снимку каждые 10 секунд.

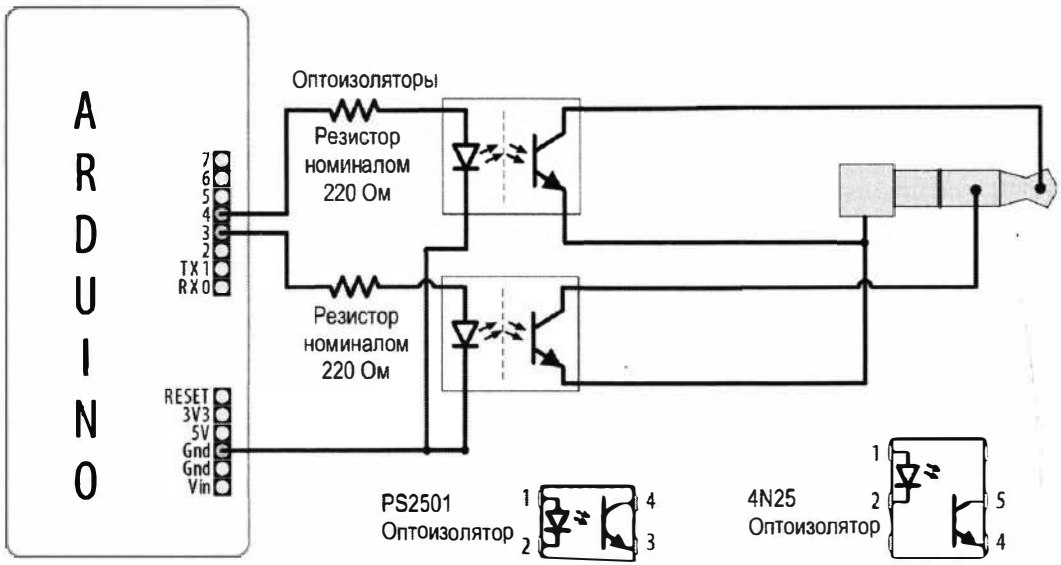


Рис. 10.3. Использование оптоизоляторов при подключении платы Arduino к кабелю с TRS-штекером

[1] См.: **https://ru.wikipedia.org/wiki/Разъём_TRS**.

Листинг 10.5. Скетч для управления фотокамерой

```
/*
* Скетч camera
* Управляет цифровой камерой, делая 20 снимков
* Контакт 4 используется для управления фокусом
* Контакт 3 используется для управления затвором
*/

int focus = 4;   // Номер контакта для подключения оптоизолятора фокуса
int shutter = 3; // Номер контакта для подключения оптоизолятора затвора

long exposure = 250;   // Выдержка затвора в миллисекундах
long interval = 10000; // Время между снимками в миллисекундах

void setup()
{
   pinMode(focus, OUTPUT);
   pinMode(shutter, OUTPUT);
   for (int i=0; i<20; i++) // Количество снимков
   {
      takePicture(exposure); // Делает снимок
      delay(interval);       // Пауза перед следующим снимком
   }
}

void loop()
{
   // Скетч завершает работу после 20 снимков,
   // поэтому функция loop() не содержит никакого кода.
   // Но ее наличие в скетче обязательно, ибо в противном случае
   // скетч не будет компилироваться
}

void takePicture(long exposureTime)
{
   int wakeup = 10; // Для камеры требуется некоторое время,
                    // чтобы выйти из режима сна и сфокусироваться
   // Эти настройки нужно откорректировать для конкретной
   // используемой камеры
   digitalWrite(focus, HIGH); // Выводим камеру из режима сна
                              // и фокусируем
   delay(wakeup); // Ожидаем, пока камера не выйдет из режима сна
                  // и не сфокусируется
   digitalWrite(shutter, HIGH); // Открываем затвор
   delay(exposureTime);         // Ждем истечения выдержки
   digitalWrite(shutter, LOW); // Отпускаем затвор
   digitalWrite(focus, LOW);   // Отпускаем фокус
}
```

Обсуждение работы решения и возможных проблем

Подключать напрямую контакты платы Arduino к контактам TRS-разъема камеры настоятельно не рекомендуется, поскольку напряжения этих двух устройств могут быть несовместимы, что может вызвать повреждение одного или обоих этих устройств. Поэтому плата Arduino подключается к камере через оптоизоляторы, которые подробно рассматриваются в *разд. 10.0*.

Чтобы подобрать правильный штекерный разъем для гнездового разъема используемой камеры, придется посмотреть требуемую информацию в руководстве камеры.

Чтобы скетч правильно функционировал, может потребоваться изменить порядок включения и выключения контактов в функции `takePicture()`. Съемка с выдержкой от руки камерами типа Canon выполняется следующим образом: нужно включить фокус, затем, удерживая фокус включенным, открыть затвор, затем закрыть его после требуемой выдержки и, наконец, выключить фокус (так это делается в скетче). Чтобы сделать снимок с выдержкой, определяемой камерой, нажимаем и отпускаем кнопку фокуса, а затем нажимаем кнопку спуска затвора.

Дополнительная информация

Управлять работой камер Canon позволяет набор Canon Hack Development (**https://oreil.ly/QcWci**) .

Подробную информацию по экспериментированию с камерами Canon вы найдете в книге «The Canon Camera Hackers Manual: Teach Your Camera New Tricks» («Руководство по камерам Canon для хакеров. Как научить свою камеру новым трюкам»), автор Berthold Daum, издательство Rocky Nook.

Камерой GoPRO можно управлять с помощью Arduino через интерфейс Wi-Fi, используя библиотеку GoPRO (**https://oreil.ly/ybd1u**).

Подобным образом можно также управлять и видеокамерами, используя интерфейс LANC. Библиотека Arduino-LANC (**https://oreil.ly/LkfXb**) обеспечивает поддержку этого интерфейса для платформы Arduino.

Существует и шилд для Arduino — Blackmagic 3G-SDI (**https://www.blackmagicdesign.com/developer/product/arduino**), позволяющий управлять видеооборудованием высокого класса Black Magic Design, включая камеры, использующие открытый протокол этой компании.

10.5. Управление электрическими устройствами с помощью модифицированного дистанционно управляемого выключателя

ЗАДАЧА

Требуется с помощью дистанционно управляемого выключателя безопасно включать и выключать сетевое напряжение для светильников и других бытовых электроприборов.

РЕШЕНИЕ

С помощью платы Arduino можно эмулировать нажатие кнопок пульта дистанционного управления выключателем сетевого напряжения устройства, используя оптоизолятор для подключения платы к пульту ДУ. Этот подход может потребоваться для пультов дистанционного управления на основе беспроводной радиосвязи.

Приведенный здесь метод можно применять практически для любого пульта дистанционного управления. Модифицирование пульта ДУ особенно предпочтительно по той причине, что изолирует потенциально опасное сетевое напряжение от платы Arduino и пользователя, поскольку подвергаемый модификации пульт ДУ работает от батарей.

Вскрытие пульта ДУ сделает недействительной гарантию на него и чревато его повреждением. Предпочтительнее использовать решения на основе ИК-сигналов, рассмотренные в этой главе, поскольку для их реализации не требуется модифицировать пульт ДУ.

Если вы хотите использовать это решение для управления выключателем, но также хотите сохранить возможность пользоваться пультом ДУ обычным способом, следует рассмотреть возможность приобретения дополнительного пульта ДУ для модификации. Большинство поставщиков будут только рады продать вам дополнительное устройство. Но при этом вам следует убедиться в том, что оно поддерживает правильную частоту для версии бытового прибора, светильника или розетки, которыми нужно управлять. Для такого дополнительного пульта ДУ может потребоваться настроить используемый им канал.

Вскройте корпус пульта ДУ и подключите к его кнопкам фототранзистор оптоизолятора (его контакты 3 и 4 на рис. 10.4), а светодиод оптоизолятора (его контакты 1 и 2) подключите к контактам платы Arduino.

В листинге 10.6 приводится скетч для управления пультом ДУ. Скетч использует сигналы от кнопок включения и выключения с самовозвратом для активирования кнопок включения и выключения на пульте дистанционного управления.

Листинг 10.6. Скетч для управления кнопками пульта ДУ

```
/*
* Скетч OptoRemote
* Управляет кнопками включения и выключения пульта дистанционного управления
* Нажатие кнопки подает на выходной контакт импульс длительностью
  минимум полсекунды
*/

const int onSwitchPin = 2;  // Контакт для подключения входного сигнала
                            / с кнопки управления кнопкой Вкл пульта ДУ
const int offSwitchPin = 3; // Контакт для подключения входного сигнала
                          // с кнопки управления кнопкой Выкл пульта ДУ
const int remoteOnPin = 4;  // Контакт для выходного сигнала
                            // для кнопки Вкл пульта ДУ
```

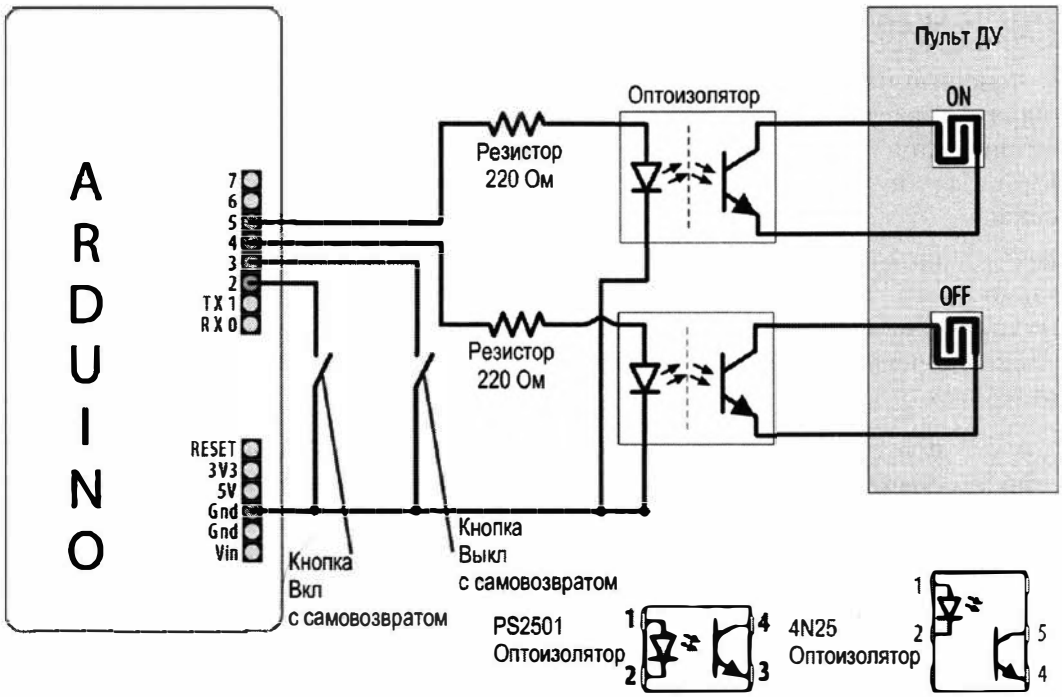


Рис. 10.4. Подключение пульта ДУ к плате Arduino через оптоизоляторы

```
const int remoteOffPin = 5; // Контакт для выходного сигнала
                            // для кнопки Выкл пульта ДУ
const int PUSHED = LOW;     // Нажатие кнопки устанавливает низкий
                            // уровень (LOW) на соответствующем контакте

void setup()
{
   pinMode(remoteOnPin, OUTPUT);
   pinMode(remoteOffPin, OUTPUT);
   pinMode(onSwitchPin, INPUT_PULLUP); // Включаем внутренние повышающие резисторы
   pinMode(offSwitchPin, INPUT_PULLUP);
}

void loop()
{
   int val = digitalRead(onSwitchPin); // Считываем входное значение
                                       // состояния кнопки включения
   // Если кнопка нажата, тогда включаем устройство, если еще не включено
   if(val == PUSHED)
   {
      pulseRemote(remoteOnPin);
   }
```

```
   val = digitalRead(offSwitchPin); // Считываем входное значение
                                    // состояния кнопки выключения
   // Если кнопка нажата, тогда выключаем устройство,
   // если еще не выключено
   if(val == PUSHED)
   {
      pulseRemote(remoteOffPin);
   }
}

// Включаем оптоизолятор на полсекунды, чтобы замкнуть кнопку пульта ДУ
void pulseRemote(int pin)
{
   digitalWrite(pin, HIGH); // Включаем оптоизолятор
   delay(500);              // Ждем полсекунды
   digitalWrite(pin, LOW);  // Выключаем оптоизолятор
}
```

Обсуждение работы решения и возможных проблем

Кнопки большинства пультов дистанционного управления состоят из ряда обнаженных пружинящих медных дорожек, которые замыкаются при нажатии находящейся над ними кнопки. Менее часто встречаются пульты ДУ с обычными кнопками. Такие пульты легче модифицировать, поскольку выводы их контактов предоставляют удобное место для припаивания проводов от оптоизолятора.

Хотя пульт ДУ с подключенной через оптоизоляторы платой Arduino можно продолжать использовать и в обычном режиме нажатия его кнопок (замыкание его контактов выполняется любым способом: нажатием собственных кнопок или активированием фототранзистора), это может быть не совсем удобным.

Транзистор в выходной цепи оптоизолятора позволяет протекание тока только в одном направлении, поэтому если первоначально замыкания цепи кнопок пульта не происходит, попробуйте поменять местами выводы подключения фототранзистора к пульту и посмотрите, решит ли это проблему.

В некоторых пультах ДУ один контакт всех кнопок подключен к общей шине (обычно к шине «земли» схемы пульта). Проверить наличие такого соединения можно, проследив подключение кнопок на плате пульта или замерив сопротивление между дорожками платы, к которым подключены разные кнопки. При нулевом сопротивлении между дорожками к каждой общей группе нужно подключать только один провод. В случае пульта ДУ небольшого размера такая организация подключения кнопок пульта облегчит подключение проводов от оптоизоляторов.

Может случиться, что каждая кнопка пульта ДУ имеет несколько контактов. В таком случае для одной кнопки может потребоваться использовать несколько оптоизоляторов, чтобы соединить все контакты. На рис. 10.5 показано подключение к одной кнопке пульта ДУ трех оптоизоляторов, управляемых одним контактом платы Arduino.

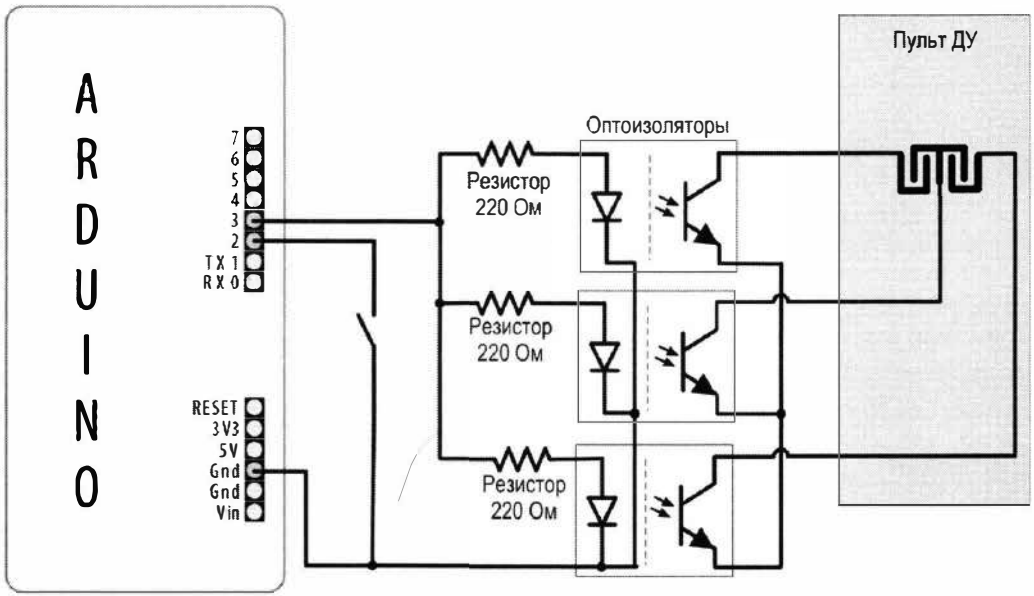


Рис. 10.5. Подключение нескольких оптоизоляторов к одной кнопке пульта ДУ

Другой подход к управлению сетевым напряжением реализуется с помощью изолирующего реле, которым можно управлять непосредственно сигналом с контакта платы Arduino. Например, силового реле Digital Loggers IoT, предлагаемого компанией Adafruit (артикул 2935) или компанией SparkFun (артикул 14236). Это реле прекрасно подойдет вместо снятого с производства реле PowerSwitch Tail.

Дополнительная информация

Оптоизоляторы используются в решении из *разд. 10.4*. Вы можете ориентироваться на него для использования их в схеме этого решения.

ГЛАВА 11

Работа с дисплеями

11.0. Введение

Жидкокристаллические (ЖК) дисплеи и дисплеи на основе органических светодиодов (OLED-дисплеи) предоставляют удобный и недорогой способ оснащения проекта пользовательским интерфейсом. В этой главе мы рассмотрим, как подключать к плате Arduino и использовать совместно с ней стандартные текстовые и графические жидкокристаллические и OLED-дисплеи. Текстовая панель на основе микросхемы HD44780 компании Hitachi, несомненно, является сейчас самым популярным жидкокристаллическим дисплеем. Она может отображать две или четыре строки текста по 16 или 20 символов в каждой строке. Существуют также 32- и 40-символьные версии этого дисплея, но обычно они значительно дороже. Среда Arduino IDE содержит библиотеку LiquidCrystal для работы с ЖКД, которая позволяет выводить текст на эти дисплеи с такой же легкостью, как и в окно монитора порта (см. *главу 4*), поскольку в ней используются те же самые базовые функции печати, что и в библиотеке Serial.

Кроме простого отображения текста жидкокристаллические дисплеи поддерживают такие специальные функции, как прокрутка или выделение отдельных слов, а также отображение специальных символов и символов иных алфавитов, чем английский.

Для текстового ЖКД можно создавать пользовательские символы и символьную графику, но он не способен отображать высококачественную графику, для которой требуется графический дисплей. Графические жидкокристаллические (ГЖКД) и OLED-дисплеи предлагаются на рынке чуть дороже, чем текстовые дисплеи.

Для подключения графического дисплея к плате Arduino может потребоваться большее количество проводов, чем в большинстве других решений, приведенных в этой книге. Неправильная коммутация является основной причиной проблем с графическими дисплеями, поэтому никогда не торопитесь при подключении этих дисплеев и всегда проверяйте и перепроверяйте правильность соединений. Большую помощь при проверке правильности подключений окажет вам недорогой мультиметр, способный измерять напряжение и сопротивление. Он сэкономит вам массу времени, ушедшего на чесание в затылке перед дисплеем, на котором ничего не отображается. Прибор этот не обязательно должен быть каким-то навороченным, поскольку и самый дешевый мультиметр поможет вам проверить наличие проводимости и правильность напряжений.

11.1. Подключение и использование текстового ЖКД

ЗАДАЧА

Требуется отображать текст и числа на ЖКД со стандартным контроллером HD44780 или совместимым с ним.

РЕШЕНИЕ

Среда Arduino IDE содержит библиотеку LiquidCrystal для управления жидкокристаллическими дисплеями с контроллером HD44780 — например, дисплеями компании SparkFun (артикул LCD-00255) или компании Adafruit (артикул 181).

Большинство жидкокристаллических дисплеев, предлагаемых для использования с платформой Arduino, будут совместимы с контроллером HD44780. В случае сомнений в совместимости используемого контроллера ЖКД с контроллером HD44780 проверьте это по его справочному листку (datasheet). Если используемый вами ЖКД оснащен контроллером на компоновочной плате, с ним, возможно, удастся организовать взаимодействие по последовательному протоколу, используя намного меньшее количество проводов. Более подробная информация на эту тему приведена в *разд 4.11*.

Для работы с ЖКД к нему необходимо подключить линии данных, управления и питания (рис. 11.1). Соедините его контакты управления и данных с цифровыми

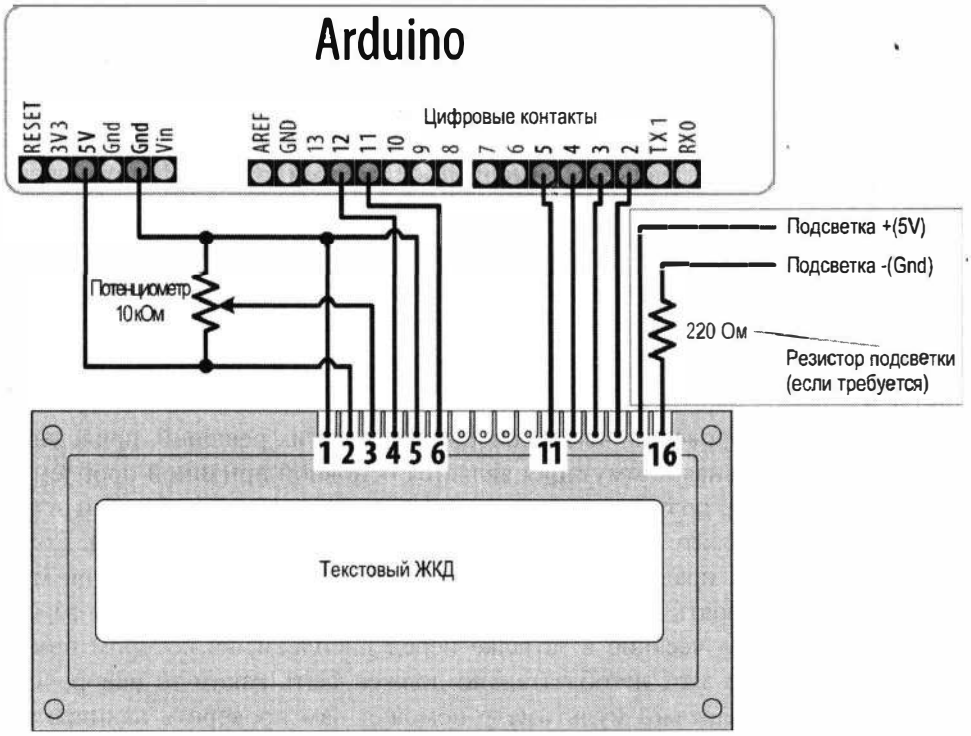


Рис. 11.1. Подключение текстового ЖКД к плате Arduino

контактами платы Arduino, подсоедините потенциометр для управления контрастом, а также подключите контакты питания. Если используемый вами дисплей оснащен фоновой подсветкой, следует подключить и контакты питания подсветки, обычно через резистор.

Не забудьте при этом свериться со справочным листком на используемый ЖКД, чтобы удостовериться в правильности подключения всех контактов.

Причина большинства проблем с ЖКД — неправильное подключение его контактов. Поэтому тщательно проверьте, что контакты платы Arduino подключены к правильным контактам ЖКД, поскольку они могут быть пронумерованы и/или размещены иначе, чем показано на рис. 11.1. Также проверьте надежность пайки соединяющих проводов или разъемов.

В табл. 11.1 приводится наиболее распространенный вариант подключения контактов ЖКД к контактам платы Arduino. Но если используемый ЖКД имеет другую схему расположения контактов, убедитесь, что его контроллер совместим с контроллером HD44780 компании Hitachi. Решение этого раздела работает только с ЖДК, чьи контроллеры совместимы с контроллером Hitachi. Жидкокристаллический дисплей должен иметь 16 выводов (14 — при отсутствии функции фоновой подсветки). Важно правильно определить контакт 1 используемого ЖКД, т. к. он может находиться в другом месте, чем показано на рис. 11.1.

Таблица 11.1. *Подключения выводов ЖКД к выводам Arduino*

Вывод ЖКД	Функция	Контакт платы Arduino
1	«Земля» (GND), или 0 В, или Vss	GND
2	+5 В или Vdd	5V
3	Vo или контраст	
4	Сброс	12
5	Чтение/запись	GND
6	Разрешение	11
7	D0	
8	D1	
9	D2	
10	D3	
11	D4	5
12	D5	4
13	D6	3
14	D7	2
15	Анод	
16	Катод	

Вам интересно, почему контакты ЖКД с 7 по 10 не подключены? Это объясняется тем, что для подачи данных на ЖКД можно использовать либо четыре, либо восемь линий. В рассматриваемом решении используется подход с использованием четырех линий, поскольку это освобождает другие четыре контакта платы Arduino для иных применений. Теоретически использование восьми линий дает лучшую производительность, но не очень существенную и не стоящую потери четырех контактов платы Arduino.

Для управления контрастом ЖКД на его контакт 3 нужно подать положительное напряжение через потенциометр. Без такого напряжения на дисплее может ничего не отображаться. На рис. 11.1 один крайний вывод потенциометра подключен к шине «земли», другой — к шине питания +5 В, а средний вывод — к контакту 3 дисплея. Питание на ЖКД подается на его контакты 1 и 2 с контактов Gnd и 5V платы Arduino соответственно.

Многие ЖКД оснащаются внутренним светодиодом для фоновой подсветки дисплея. Информация о наличии фоновой подсветки в используемом вами ЖКД, а также о том, требуется ли для ее питания внешний резистор, чтобы не допустить перегорания светодиода подсветки, должна присутствовать в справочном листке дисплея. При наличии любых сомнений в необходимости такого резистора следует не рисковать и использовать резистор номиналом 220 Ом. Выводы подсветки поляризованы, поэтому контакт 15 ЖКД нужно подключить к контакту +5V платы Arduino, а контакт 16 — к контакту Gnd (на рис. 11.1 резистор подсветки подключен между контактом 16 ЖКД и контактом Gnd платы Arduino, но его с таким же успехом можно подключить между контактом 15 ЖКД и контактом 5V платы Arduino).

Прежде чем подключать питание, внимательно проверьте и перепроверьте правильность подключение дисплея, т. к. подключение питания с неправильной полярностью может вывести ЖКД из строя. Убедившись в правильности подключения ЖКД, проверьте его работоспособность — загрузите в плату Arduino скетч примера, выполнив команду меню среды Arduino IDE **Файл | Примеры | LiquidCrystal | HelloWorld**.

В листинге 11.1 приводится слегка модифицированный код этого примера. Если необходимо, откорректируйте значения констант `numRows` и `numCols` согласно соответствующим характеристикам используемого дисплея.

Листинг 11.1. Код примера HelloWorld из библиотеки LiquidCrystal

```
/*
* Скетч LiquidCrystal Library - Hello World
*
* Демонстрирует работу ЖКД 16 × 2
* https://www.arduino.cc/en/Tutorial/HelloWorld
*/

#include <LiquidCrystal.h> // Подключаем библиотеку LiquidCrystal
// Константы для количества строк и столбцов ЖКД
```

```
const int numRows = 2;
const int numCols = 16;

// Инициализируем библиотеку номерами линий интерфейса
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
   lcd.begin(numCols, numRows);
   lcd.print("Hello, World!"); // Выводим текстовую строку на ЖКД
}

void loop()
{
   // Устанавливаем позицию курсора на столбец 0, линию 1
   // (Нумерация строк начинается с 0, поэтому строка 1 является второй строкой):
   lcd.setCursor(0, 1);
   // Выводим на дисплей обновляющееся количество секунд
   // исполнения скетча
   lcd.print(millis()/1000);
}
```

После загрузки скетча в плату Arduino в первой строке ЖКД должен отобразиться текст **Hello, World!**, а во второй строке — число, увеличивающееся на единицу каждую секунду.

Обсуждение работы решения и возможных проблем

Если вы абсолютно уверены в правильности подключения дисплея, но на нем ничего не отображается, возможно, следует установить необходимый уровень контраста с помощью соответствующего потенциометра. Поворот вала потенциометра до максимума в одном направлении (обычно в сторону, подключенную к «земле») устанавливает максимальный контраст, когда в позициях символов отображаются затененные прямоугольные блоки. А в результате поворота вала до упора в обратном направлении на дисплее, скорее всего, отображаться ничего не будет. Правильная настройка контраста зависит от многих факторов, включая угол обзора и температуру окружающей среды. Поэтому просто вращайте вал потенциометра настройки контраста, пока не добьетесь наилучшего возможного отображения.

Если при любой настройке потенциометра на дисплее не отображаются затененные блоки пикселов, проверьте правильность подключения линий сигналов управления к контактам дисплея.

Единожды добившись отображения текста на ЖКД, вы можете быть уверены, что и дальнейшее использование его в скетчах не будет представлять для вас никаких проблем. Для вывода текста на дисплей используются те же самые команды, что и для вывода в окно монитора порта по последовательному интерфейсу (см. *главу 4*).

В решении, приведенном в следующем разделе, рассматриваются эти команды, а также показано, как управлять расположением текста на дисплее.

Дополнительная информация

Дополнительная информация по работе с библиотекой LiquidCrystal приводится на ее веб-странице справки сайта Arduino (**https://oreil.ly/J_UEr**).

Подробная информация по команде `print()` приведена в *главе 4*.

Справочный листок (datasheet) на контроллер ЖКД HD44780 содержит подробную справку о его низкоуровневой функциональности. Библиотека LiquidCrystal позволяет не беспокоиться на этот счет в большинстве случаев работы с контроллером, скрывая от пользователя все такие подробности, но они могут быть полезны для тех, кто хочет знать о базовых возможностях микроконтроллера. Загрузить этот справочный листок можно здесь: **https://oreil.ly/xgocv**.

На странице LCD Displays (**https://oreil.ly/Sz3Hb**) веб-сайта Arduino Playground предлагается большой выбор программного обеспечения и множество полезной информации по работе с жидкокристаллическими дисплеями, а также приводятся ссылки на другие соответствующие ресурсы.

11.2. Форматирование выводимого текста

ЗАДАЧА

Требуется управлять размещением выводимого текста на экране ЖКД. Например, чтобы отображать числа в определенных позициях экрана.

РЕШЕНИЕ

Решение этой задачи предлагается в скетче из листинга 11.2. Скетч отображает на ЖКД убывающий счет от 9 до 0, после чего выводит на экран последовательность цифр в трех четырехсимвольных столбцах. Если необходимо, откорректируйте значения констант `numRows` и `numCols` согласно соответствующим характеристикам используемого дисплея.

Листинг 11.2. Отображение на дисплее форматированного текста

```
/*
* Скетч LiquidCrystal Library - FormatText

*/

#include <LiquidCrystal.h> // Подключаем библиотеку LiquidCrystal

// Константы для количества строк и столбцов ЖКД
const int numRows = 2;
const int numCols = 16;
```

```
int count;

// Инициализируем библиотеку номерами линий интерфейса
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
   lcd.begin(numCols, numRows);
   lcd.print("Starting in "); // Строка содержит 12 символов
   for(int i=9; i > 0; i--)   // Считаем обратно от 9
   {
      // Строка 0 является верхней строкой дисплея
      lcd.setCursor(12,0); // Перемещаем курсор в конец строки
      lcd.print(i);
      delay(1000);
   }
}

void loop()
{
   int columnWidth = 4;     // Количество цифр в выводимом столбце
   int displayColumns = 3; // Количество столбцов цифр
   lcd.clear();
   for( int col=0; col < displayColumns; col++)
   {
      lcd.setCursor(col * columnWidth, 0);
      count = count+ 1;
      lcd.print(count);
   }
   delay(1000);
}
```

Обсуждение работы решения и возможных проблем

Функция `print()` (и ее разновидности) библиотеки LiquidCrystal похожа на эту функцию библиотеки Serial. Кроме того, библиотека LiquidCrystal еще содержит команды для управления размещением курсора на экране дисплея (указание строки и столбца для отображения выводимого символа).

Оператор `lcd.print()` последовательно отображает на экране символы один за другим. Текст, превышающий длину строки экрана, может не отображаться или же отображаться на другой строке экрана. Функция `lcd.setCursor()` позволяет указывать на экране начальную позицию текста, выводимого функцией `lcd.print()`. Этой функции передаются два параметра, указывающие столбец и строку экрана (верхний левый угол экрана имеет координаты 0,0). Функция `lcd.print()` начнет выводить текст в позиции, указанной функцией `lcd.setCursor()`, затирая существующий текст. Скетч приводимого решения использует эту функцию для вывода чисел в определенных позициях экрана.

Например, в следующем фрагменте кода в функции `setup()`:

```
lcd.setCursor(12,0); // перемещаем курсор в 13-ю позицию
lcd.print(i);
```

функция `lcd.setCursor(12,0)` обеспечивает вывод каждого числа в одной и той же позиции экрана — в тринадцатом столбце второй строки. В результате числа отображаются в фиксированной позиции экрана, а не следуют на экране одно за другим.

Нумерация строк и столбцов экрана ЖКД начинается с нуля, поэтому команда, например `setCursor(4,0)` задает позицию в пятом столбце первой строки экрана.

В следующем фрагменте кода функция `setCursor()` распределяет начало каждого столбца, задавая начало следующего столбца от начала предыдущего на расстоянии значения переменной `columnWidth`:

```
lcd.setCursor(col * columnWidth, 0);
count = count+ 1;
lcd.print(count);
```

А функция `lcd.clear()` очищает экран и перемещает курсор в левый верхний угол.

В листинге 11.3 приводится вариант кода `loop()`, который отображает числа на всех строках экрана ЖКД. Просто замените код функции `loop()` в скетче из листинга 11.2 этим кодом. При этом, если необходимо, откорректируйте значения констант `numRows` и `numCols` согласно соответствующим характеристикам используемого дисплея.

Листинг 11.3. Вывод чисел на всех строках экрана ЖКД

```
void loop()
{
   int columnWidth = 4;
   int displayColumns = 3;
   lcd.clear();
   for(int row=0; row < numRows; row++)
   {
      for( int col=0; col < displayColumns; col++)
      {
         lcd.setCursor(col * columnWidth, row);
         count = count+ 1;
         lcd.print(count);
      }
   }
   delay(1000);
}
```

Первый цикл `for` обрабатывает доступные строки, а второй — столбцы.

Для корректировки количества отображаемых в строке чисел, чтобы они помещались в строку, значение переменной `displayColums` следует вычислять, а не задавать. Для этого замените эту строку кода:

```
int displayColumns = 3;
```

следующей:

```
int displayColumns = numCols / columnWidth;
```

Дополнительная информация

Дополнительная информация по работе с библиотекой LiquidCrystal приводится на ее веб-странице справки сайта Arduino (**https://oreil.ly/aBU2C**).

11.3. Включение и выключение курсора и дисплея

ЗАДАЧА

Требуется мигать курсором ЖКД, а также всем дисплеем (т. е. регулярно включать и выключать дисплей). Такая функциональность может быть полезной, например, для привлечения внимания к определенной области экрана дисплея.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 11.4. Скетч показывает, как реализовать мигание курсора. Под миганием курсора имеется в виду включающийся и выключающийся блок в позиции вывода следующего символа. В скетче также показано мигание дисплеем — чтобы, например, привлечь внимание пользователя.

Листинг 11.4. Мигание курсором и дисплеем

```
/*
* Скетч Cursor blink
*/

// Подключаем библиотеку LiquidCrystal
#include <LiquidCrystal.h>

// Инициализируем библиотеку номерами линий интерфейса
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
   // Задаем количество столбцов и строк ЖК-дисплея:
   lcd.begin(16, 2);
   // Выводим текстовую строку на ЖКД
   lcd.print("hello, world!");
}
```

```
void loop()
{
   lcd.setCursor(0, 1);
   lcd.print("cursor blink");
   lcd.blink();
   delay(2000);
   lcd.noBlink();
   lcd.print(" noBlink");
   delay(2000);
   lcd.clear();
   lcd.print("Display off ...");
   delay(1000);
   lcd.noDisplay();
   delay(2000);
   lcd.display(); // Включаем дисплей обратно
   lcd.setCursor(0, 0);
   lcd.print(" display flash !");
   displayBlink(2, 250); // Двойное мигание
   displayBlink(2, 500); // Опять двойное мигание в течение вдвое большего периода
   lcd.clear();
}

void displayBlink(int blinks, int duration)
{
   while(blinks--)
   {
      lcd.noDisplay();
      delay(duration);
      lcd.display();
      delay(duration);
   }
}
```

Обсуждение работы решения и возможных проблем

Для включения и выключения мигания курсора в скетче вызываются функции `blink()` и `noBlink()` соответственно.

Возможность мигания всем дисплеем реализуется функцией `displayBlink()`, которая мигает экраном дисплея указанное количество раз. Эта функция использует функции `lcd.display()` и `lcd.noDisplay()` для включения и выключения экрана дисплея (при этом содержимое внутренней памяти экрана не стирается).

11.4. Прокрутка текста

ЗАДАЧА

Требуется выполнять прокрутку текста на экране ЖКД. Например, чтобы создать перемещающуюся строку, способную отображать большее количество символов, чем может поместиться в одну статическую строку экрана ЖКД.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 11.5. В скетче показано использование функций: `lcd.ScrollDisplayLeft()` и `lcd.ScrollDisplay Right()`.

Скетч прокручивает строку текста справа налево при наклоне платы Arduino, оснащенной датчиком наклона, и слева направо при отсутствии наклона. Датчик наклона подключается одним выводом к контакту 7 платы Arduino, а другим — к контакту GND. Работа с датчиком наклона подробно рассматривается в *разд. 6.2*.

Листинг 11.5. Прокрутка текста на экране ЖКД

```
/*
* Скетч Scroll
* Прокручивает текст справа налево при наклоне,
* и слева направо при отсутствии наклона
*/

#include <LiquidCrystal.h>

// Инициализируем библиотеку номерами линий интерфейса
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int numRows = 2;
const int numCols = 16;
const int tiltPin = 7;    // Номер контакта для подключения датчика наклона
const char textString[] = "tilt to scroll"; // Наклоните для прокрутки текста
const int textLen = sizeof(textString) - 1; // Количество символов

bool isTilted = false;

void setup()
{
   // Задаем количество столбцов и строк ЖК-дисплея:
   lcd.begin(numCols, numRows);
   pinMode(tiltPin, INPUT_PULLUP);
   lcd.print(textString);
}
```

```
void loop()
{
   if(digitalRead(tiltPin) == LOW && isTilted == false)
   {
      // Переходим сюда при наклоне и прокручиваем текст справа налево
      isTilted = true;
      for (int position = 0; position < textLen; position++)
      {
         lcd.scrollDisplayLeft();
         delay(150);
      }
   }
   if(digitalRead(tiltPin) == HIGH && isTilted == true)
   {
      // Переходим сюда, если нет наклона, поэтому прокручиваем
      // слева направо
      isTilted = false;
      for (int position = 0; position < textLen; position++)
      {
         lcd.scrollDisplayRight();
         delay(150);
      }
   }
}
```

Обсуждение работы решения и возможных проблем

Первая половина главного цикла `loop()` обрабатывает изменение состояния из не наклоненного в наклоненное. Для этого код проверяет, замкнут ли датчик наклона (сигнал низкого уровня — `LOW`), или разомкнут (сигнал высокого уровня — `HIGH`). Если датчик замкнут, и его текущее состояние (хранящееся в переменной `isTilted`) не наклоненное, код начинает прокручивать текст справа налево. Скорость прокрутки управляется паузой в цикле `for`, длительность которой можно корректировать, чтобы ускорить или замедлить скорость.

А вторая половина цикла `loop()` обрабатывает обратное изменение состояния — из наклоненного в не наклоненное.

Возможность прокрутки текста особенно полезна в тех случаях, когда длина отображаемого текста превышает длину строки экрана ЖКД. Эта возможность демонстрируется скетчем из листинга 11.6. Скетч использует функцию `marquee()`, которая может прокручивать текст длиной до 32 символов.

Листинг 11.6. Прокрутка текста длиной до 32 символов

```
/*
* Скетч Marquee
* Прокручивает очень длинную строку текста на экране ЖКД
*/
```

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int numRows = 2;
const int numCols = 16;

void setup()
{
   // Задаем количество столбцов и строк ЖК-дисплея:
   lcd.begin(numCols, numRows);
   marquee("This is a very long string of text that will scroll");
      // Это очень длинная строка текста, которая будет прокручиваться
}

void loop()
{
   // Здесь ничего не делаем.
}

// Эта функция использует прокрутку для отображения сообщения
// длиной до 32 символов
void marquee(char *text)
{
   lcd.print(text);
   delay(1000);
   for (int position = 0; position < strlen(text)-numCols; position++)
   {
      lcd.scrollDisplayLeft();
      delay(300);
   }
}
```

Скетч использует функцию `lcd.scrollDisplayLeft()` для прокрутки отображаемого текста, если длина текста превышает длину экрана ЖКД.

Контроллер ЖКД оснащен памятью, в которой хранится отображаемый текст. Но объем этой памяти не очень большой — обычно 32 байта для большинства четырехстрочных дисплеев. В случае более длинных сообщений их последние символы могут записываться в памяти поверх первых символов, таким образом затирая их. Для отображения прокруткой более длинных сообщений (например, твитов) или для более точного управления прокруткой требуется иной подход. Функция в листинге 11.7 сохраняет текст сообщения в памяти RAM платы Arduino и отправляет его по частям на экран дисплея, чтобы создать эффект прокрутки. Сообщения могут быть любой длины, допустимой памятью платы Arduino.

Листинг 11.7. Функция `marquee()` для отображения прокруткой сообщений любой длины

```
// Эта версия функции marquee() использует ручную прокрутку
// для отображения очень длинных сообщений
void marquee(char *text)
{
   int length = strlen(text); // Количество символов в тексте сообщения
   if(length < numCols)
   lcd.print(text);
   else
   {
      int pos;
      for(pos = 0; pos < numCols; pos++)
      lcd.print(text[pos]);
      delay(1000); // Даем время для чтения первой строки,
                   // прежде чем начинать прокрутку
      pos=1;
      while(pos <= length - numCols)
      {
         lcd.setCursor(0,0);
         for(int i=0; i < numCols; i++)
         lcd.print(text[pos+i]);
         delay(300);
         pos = pos + 1;
      }
   }
}
```

11.5. Отображения на экране ЖКД специальных символов

ЗАДАЧА

Требуется отображать на экране ЖКД специальные символы: ° (градусы), π (пи) или любой другой символ, хранящийся в памяти ЖКД.

РЕШЕНИЕ

Для отображения требуемого специального символа нужно определить его код по таблице шаблонов символов в справочном листке на используемый ЖКД, а затем отобразить его с помощью функции `write()`. В листинге 11.8 приводится код скетча, который в функции `setup()` настраивает отображение некоторых наиболее распространенных специальных символов, а затем в функции `loop()` выводит их на экран.

Листинг 11.8. Отображает специальные символы

```
/*
* Скетч LiquidCrystal Library - Special Chars
*/

#include <LiquidCrystal.h>

//Константы для количества строк и столбцов ЖКД
const int numRows = 2;
const int numCols = 16;

// Константы для некоторых полезных символов
const byte degreeSymbol = B11011111;
const byte piSymbol = B11110111;
const byte centsSymbol = B11101100;
const byte sqrtSymbol = B11101000;
const byte omegaSymbol = B11110100; // Символ для единицы Ом
byte charCode = 32; // Первый отображаемый символ ASCII

int col;
int row;

// Инициализируем библиотеку номерами линий интерфейса
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
   lcd.begin(numRows, numCols);
   showSymbol(degreeSymbol, "degrees");
   showSymbol(piSymbol, "pi");
   showSymbol(centsSymbol, "cents");
   showSymbol(sqrtSymbol, "sqrt");
   showSymbol(omegaSymbol, "ohms");
   lcd.clear();
}

void loop()
{
   lcd.write(charCode);
   calculatePosition();
   if(charCode == 255)
   {
      // Завершили вывод всех символов. Ждем десять секунд и повторяем процесс
      delay(2000);
      lcd.clear();
      row = col = 0;
      charCode = 32;
   }
```

```
    charCode = charCode + 1;
}

void calculatePosition()
{
    col = col + 1;
    if( col == numCols)
    {
        col = 0;
        row = row + 1;
        if( row == numRows)
        {
            row = 0;
            delay(2000); // Пауза
            lcd.clear();
        }
        lcd.setCursor(col, row);
    }
}

// Функция для отображения символа и его описания
void showSymbol(byte symbol, char *description)
{
    lcd.clear();
    lcd.write(symbol);
    lcd.print(' '); // Вставляем пробел перед описанием
    lcd.print(description);
    delay(3000);
}
```

Обсуждение работы решения и возможных проблем

Справочный листок на контроллер ЖКД (**https://oreil.ly/nv0ZJ**) содержит таблицу шаблонов всех возможных символов.

Найдите в этой таблице символ, который требуется отобразить. Код этого символа определяется комбинацией двоичных значений его столбца и строки, как показано на рис. 11.2.

Например, знак градуса (°) находится в третьей с конца позиции нижней строки таблицы, показанной на рис. 11.2. Заголовок этого столбца содержит старшие четыре бита (`1101`), а заголовок строки — младшие четыре (`1111`). Конкатенируя эти коды, получим полный код для этого символа: `B11011111`. В скетче можно использовать как это двоичное значение, так и его эквивалентное шестнадцатеричное (`0xDF`) или десятичное значение (`223`). Обратите внимание, что на рис. 11.2 показаны только четыре из 16 строк таблицы шаблонов символов из справочного листка.

Для отображения на ЖК-дисплее любых символов ASCII, указанных в таблице, можно также использовать функцию `lcd.write()`, передавая ей код ASCII символа.

Рис. 11.2. Фрагмент таблицы шаблонов символов из справочного листка (верхние четыре строки)

В скетче решения также используется функция `showSymbol()` для отображения символа и его описания:

```
void showSymbol(byte symbol, char *description)
```

Подробная информация по работе со строками символов и передаче их функциям в качестве параметров приведена в *разд. 2.6*.

Дополнительная информация

Подробная информация по контроллеру жидкокристаллического дисплея HD44780 компании Hitachi приводится в справочном листке на него (**https://oreil.ly/xu4li**).

11.6. Создание пользовательских символов

ЗАДАЧА

Требуется определить символы, не входящие в состав таблицы шаблонов стандартных символов контроллера ЖКД, а затем отображать эти символы на дисплее.

РЕШЕНИЕ

Скетч примера решения этой задачи приводится в листинге 11.9. Скетч создает анимацию лица, выражение которого меняется между улыбающимся и хмурым.

Листинг 11.9. Пример создания и отображения пользовательского символа

```
/*
* Скетч custom_char
* Создает анимацию лица с помощью пользовательских символов
*/
```

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte happy[8] =
{
   B00000,
   B10001,
   B00000,
   B00000,
   B10001,
   B01110,
   B00000,
   B00000
};

byte saddy[8] =
{
   B00000,
   B10001,
   B00000,
   B00000,
   B01110,
   B10001,
   B00000,
   B00000
};

void setup()
{
   lcd.createChar(0, happy); // Улыбающийся
   lcd.createChar(1, saddy); // Хмурый
   lcd.begin(16, 2);
}

void loop()
{
   for (int i=0; i<2; i++)
   {
      lcd.setCursor(0,0);
      lcd.write(i);
      delay(500);
   }
}
```

Обсуждение работы решения и возможных проблем

Библиотека LiquidCrystal позволяет создать до восьми пользовательских символов, которым присваиваются коды от 0 до 8. Каждый пользовательский символ отобра-

жается на экране в матрице размером 5×8 пикселов. Для определения символа нужно создать массив из восьми байтов, каждый из которых определяет состояние пикселов соответствующей строки матрицы. При записи значения байта в виде двоичного числа значение бита 1 означает включенный соответствующий пиксел, а 0 — выключенный. Биты, начиная с шестого, игнорируются. В скетче решения создают два пользовательских символа: `happy` (улыбающийся) и `saddy` (хмурый) (рис. 11.3).

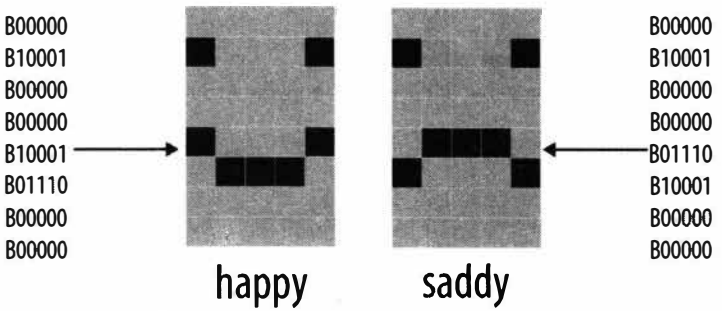


Рис. 11.3. Определение двух пользовательских символов

Символ `happy` создается в функции `setup()` на основе данных массива `happy` с помощью следующей строки кода:

```
lcd.createChar(0, happy);
```

Созданному таким образом пользовательскому символу `happy` присваивается код 0. На экран дисплея этот символ выводится следующей строкой кода:

```
lcd.write(0);
```

Код в цикле `loop()` попеременно выводит то символ 0 (`happy`), то символ 1 (`saddy`), создавая таким образом простую анимацию.

Обратите внимание на разные результаты вывода при указании параметра 0 в одинарных кавычках и без таковых. В частности, следующая строка кода отобразит число 0, а не символ `happy`:

```
lcd.write('0'); // Отображает число 0
```

11.7. Отображение символов большего размера, чем стандартные

ЗАДАЧА

Требуется объединить два или больше пользовательских символа, чтобы создать один пользовательский символ большего размера, чем стандартный. Например, создать числа высотой вдвое большей, чем стандартные.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 11.10. Скетч выводит на экран ЖКД числа вдвое более высокие, чем обычные, используя пользовательские символы.

Листинг 11.10. Создание и отображение пользовательских символов большого размера

```
/*
* Скетч customChars
* Отображает на ЖКД цифры двойной высоты
* Массивы bigDigit были предложены членом dcb форума Arduino
*/

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte glyphs[5][8] =
{
   { B11111,B11111,B00000,B00000,B00000,B00000,B00000,B00000 },
   { B00000,B00000,B00000,B00000,B00000,B00000,B11111,B11111 },
   { B11111,B11111,B00000,B00000,B00000,B00000,B11111,B11111 },
   { B11111,B11111,B11111,B11111,B11111,B11111,B11111,B11111 } ,
   { B00000,B00000,B00000,B00000,B00000,B01110,B01110,B01110 }
};

const int digitWidth = 3; // Ширина большой цифры в стандартных символах
                          // (не включая пробелы между символами)

// Массивы указателей на пользовательские символы, из которых
// будут состоять большие цифры
// Цифры 0-4                                  0      1       2      3      4
const char bigDigitsTop[10][digitWidth]={ 3,0,3, 0,3,32, 2,2,3, 0,2,3, 3,1,3,
// Цифры 5-9                                 5      6      7      8      9
                                          3,2,2, 3,2,2, 0,0,3, 3,2,3, 3,2,3};

const char bigDigitsBot[10][ digitWidth]={ 3,1,3, 1,3,1, 3,1,1, 1,1,3, 32,32,3,
                                           1,1,3, 3,1,3, 32,32,3, 3,1,3, 1,1,3};

char buffer[12]; // Используется для преобразования числа в строку

void setup ()
{
   lcd.begin(20,4);
   // Создаем пользовательские символы
   for(int i=0; i < 5; i++)
   lcd.createChar(i, glyphs[i]); // Создаем 5 пользовательских символов
```

```
   // Отображаем обратный счетчик
   for(int digit = 9; digit >= 0; digit--)
   {
      showDigit(digit, 0); // Выводим цифру
      delay(1000);
   }
   lcd.clear();
}

void loop ()
{
   // Выводим на дисплей обновляющееся количество секунд
   // исполнения скетча
   int number = millis() / 1000;
   showNumber(number, 0);
   delay(1000);
   Serial.begin(9600);
}

void showDigit(int digit, int position)
{
   lcd.setCursor(position * (digitWidth + 1), 0);
   for(int i=0; i < digitWidth; i++)
   lcd.write(bigDigitsTop[digit][i]);
   lcd.setCursor(position * (digitWidth + 1), 1);
   for(int i=0; i < digitWidth; i++)
   lcd.write(bigDigitsBot[digit][i]);
}

void showNumber(int value, int position)
{
   int index; // Указатель выводимой цифры. Самая левая цифра
              // находится в позиции 0
   String valStr = String(value);

   // Последовательно отображаем все цифры
   for(index = 0; index < 5; index++) // Отображаем до пяти цифр
   {
      char c = valStr.charAt(index);
      if(c == 0) // Проверка на символ null (не то же самое, что '0')
      return;     // Конец строки обозначается символом null
      c = c - 48; // Преобразовываем значение кода ASCII для цифры
                  // в соответствующее числовое значение (см. главу 2)
      showDigit(c, position + index);
   }
}
```

Обсуждение работы решения и возможных проблем

В обычном режиме работы ЖКД отображает символы стандартного размера, но, объединяя стандартные символы, можно создавать символы большего размера. Скетч этого решения создает пять пользовательских символов, используя подход, рассмотренный в *разд. 11.6*. Эти символы (рис. 11.4) можно объединять, чтобы создать символы двойного размера (рис. 11.5). Скетч отображает на ЖКД убывающий счет от 9 до 0, используя цифры большого размера. Затем на дисплей выводится обновляющееся количество секунд исполнения скетча.

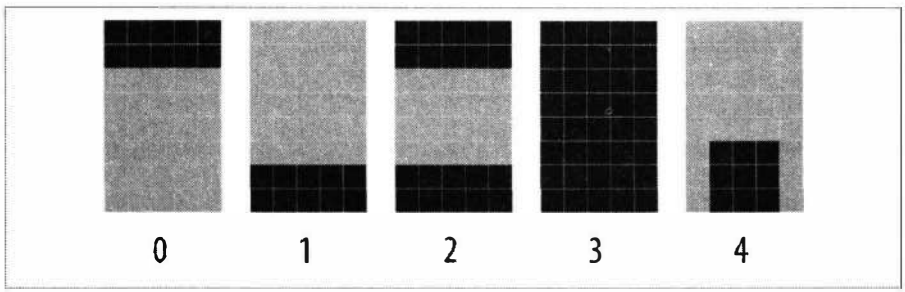


Рис. 11.4. Пять пользовательских символов, используемых для создания цифр крупного размера

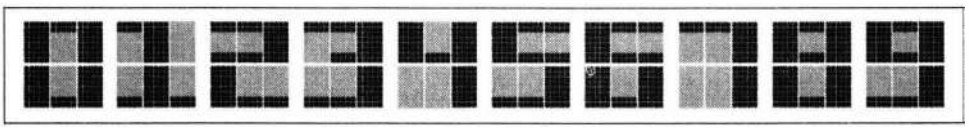


Рис. 11.5. Десять цифр большого размера, созданные объединением пользовательских символов

Массив `glyphs` используется для хранения определений пикселов для пяти пользовательских символов. Это двумерный массив с размерностями, указанными в квадратных скобках:

```
byte glyphs[5][8] = {
```

Размерность `[5]` определяется количеством пользовательских символов, а `[8]` — количеством строк в каждом таком символе. Каждый элемент массива содержит комбинации битов, значения которых (1 или 0) обозначают, включен или выключен соответствующий пиксел в этой строке. Если сравнить значения битов элемента массива `glyph[0]` (первый пользовательский символ) с первым символом на рис. 11.5, то можно увидеть, что значения 1 соответствуют темным пикселам:

```
{ B11111,B11111,B00000,B00000,B00000,B00000,B00000,B00000 } ,
```

Каждая цифра крупного размера составляется из шести пользовательских символов, три из которых формируют ее верхнюю часть, а три — нижнюю. Массивы `bigDigitsTop` и `bigDigitsBot` определяют пользовательские символы, используемые для верхних и нижних строк на экране ЖКД.

Дополнительная информация

Для отображения цифр еще большего размера можно использовать 7-сегментные светодиодные дисплеи, которые рассматриваются в *главе 7*. На рынке предлагаются 7-сегментные дисплеи с размером цифр от 12 до 50 мм и даже больше. Такие дисплеи потребляют больший ток, чем жидкокристаллические дисплеи, и не могут особо похвастаться качеством отображаемых на них символов, но они будут хорошим выбором, когда нужно отображать особо крупные символы.

11.8. Отображение символов меньшего размера, чем стандартные

ЗАДАЧА

Требуется отображать на ЖКД информацию с более высоким разрешением, чем стандартные символы, — например, столбчатую диаграмму.

РЕШЕНИЕ

В *разд. 11.7* рассматривается создание символов крупного размера объединением пользовательских символов стандартного размера. Здесь же пользовательские символы используются для решения обратной задачи. В частности, мы создадим восемь пользовательских символов малого размера. При этом высота каждого следующего символа на пиксел больше, чем предыдущего (рис. 11.6).

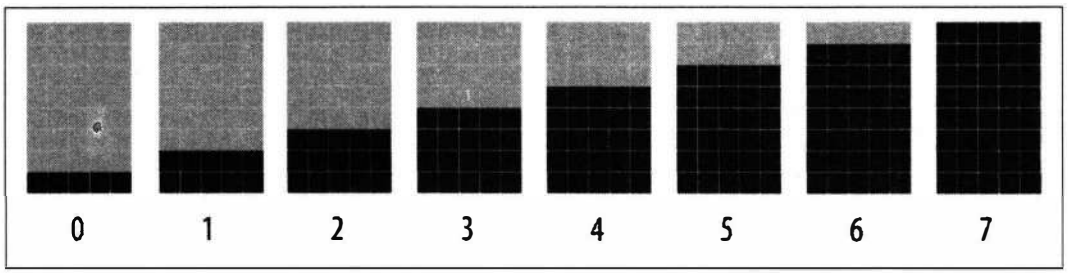


Рис. 11.6. Восемь пользовательских символов, используемых для создания вертикальных столбцов

Эти символы в скетче из листинга 11.11 используются для отображения на экране ЖКД столбчатых диаграмм.

Листинг 11.11. Отображение столбчатых диаграмм с помощью пользовательских символов малого размера

```
/*
* Скетч customCharPixels
*/

#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

//Константы для количества строк и столбцов ЖКД
const int numRows = 2;
const int numCols = 16;

// Массивы битов, определяющих состояния пикселов для восьми пользовательских символов
// 1 и 0 указывают включенное и выключенное состояние соответственно
byte glyphs[8][8] = {
      {B00000,B00000,B00000,B00000,B00000,B00000,B00000,B11111},  // 0
      {B00000,B00000,B00000,B00000,B00000,B00000,B11111,B11111},  // 1
      {B00000,B00000,B00000,B00000,B00000,B11111,B11111,B11111},  // 2
      {B00000,B00000,B00000,B00000,B11111,B11111,B11111,B11111},  // 3
      {B00000,B00000,B00000,B11111,B11111,B11111,B11111,B11111},  // 4
      {B00000,B00000,B11111,B11111,B11111,B11111,B11111,B11111},  // 5
      {B00000,B11111,B11111,B11111,B11111,B11111,B11111,B11111},  // 6
      {B11111,B11111,B11111,B11111,B11111,B11111,B11111,B11111}}; // 7

void setup ()
{
   lcd.begin(numCols, numRows);
   for(int i=0; i < 8; i++)
   lcd.createChar(i, glyphs[i]); // Создаем пользовательские символы
   lcd.clear();
}

void loop ()
{
   for( byte i=0; i < 8; i++)
   lcd.write(i); // Отображаем все восемь столбцов одной высоты
   delay(2000);
   lcd.clear();
}
```

Обсуждение работы решения и возможных проблем

В скетче создается восемь пользовательских символов малого размера. При этом высота каждого следующего символа на пиксел больше, чем предыдущего (см. рис. 11.6). Эти символы последовательно отображаются в верхней строке экрана ЖКД. Символы такой как бы «столбчатой диаграммы» можно использовать для отображения значений, масштабированных к диапазону значений от 0 до 7. Например, следующий фрагмент кода отображает значение, считываемое с аналогового ввода 0:

```
int value = analogRead(A0);
byte glyph = map(value, 0,1023, 0,8); // пропорциональное значение
                                      // в диапазоне от 0 до 7
```

```
lcd.write(glyph);
delay(100);
```

Символы столбцов можно отображать в двух строках дисплея, чтобы получить более высокое разрешение. В листинге 11.12 приводится код функции `doubleHeighBars()`, которая отображает в виде столбчатой диаграммы с вертикальным разрешением в 16 пикселов значения от 0 до 15, используя для этого две строки дисплея.

Листинг 11.12. Использование функции `doubleHeightBars()` для отображения столбчатой диаграммы с повышенным вертикальным разрешением

```
void doubleHeightBars(int value, int column)
{
   char upperGlyph;
   char lowerGlyph;
   if(value < 8)
   {
      upperGlyph = ' '; // Все пикселы выключены
      lowerGlyph = value;
   }
   else
   {
      upperGlyph = value - 8;
      lowerGlyph = 7; // Все пикселы включены
   }

   lcd.setCursor(column, 0); // Отображаем верхнюю половину
   lcd.write(upperGlyph);
   lcd.setCursor(column, 1); // Отображаем нижнюю половину
   lcd.write(lowerGlyph);
}
```

Функцию `doubleHeightBars()` можно также использовать для отображения в виде столбчатой диаграммы значения аналогового ввода, как показано в листинге 11.13.

Листинг 11.13. Использование функции `doubleHeightBars()` для отображения в виде столбчатой диаграммы значения аналогового ввода

```
for(int i=0; i < 16; i++)
{
   int value = analogRead(A0);
   value = map(value, 0, 1023, 0,16);
   doubleHeightBars(value, i); // Отображаем значение в диапазоне от 0 до 15
   delay(1000); // Интервал в 1 секунду между считываниями входящего сигнала
}
```

Подобно вертикальным столбцам можно создавать и горизонтальные полосы. Для этого нужно определить пять пользовательских символов, каждый следующий из

которых на один пиксел шире предыдущего, и использовать для создания выводимого на экран символа логику, аналогичную логике для вертикальных полос.

11.9. Выбор графического жидкокристаллического дисплея

ЗАДАЧА

Требуется отображать графику или текст на монохромном или цветном жидкокристаллическом дисплее.

РЕШЕНИЕ

На рынке предлагается большое количество графических дисплеев, которые можно использовать с платой Arduino. При выборе дисплея следует руководствоваться следующими главными его характеристиками: разрешение (количество пикселов и строк текста), размер дисплея, сенсорная возможность, объем встроенной SD-памяти и стоимость. Кроме этого, также нужно подобрать библиотеку для взаимодействия с выбранным дисплеем, что может оказаться задачей не из легких из-за большого числа таких библиотек. В этом разделе дается обзор предлагаемых дисплеев и библиотек для них, а также приводятся ссылки на дополнительную информацию, которая может быть полезной при выборе правильного дисплея для своего проекта.

Все ускоряющееся появление новых моделей смартфонов и сходных с ними устройств вылилось в снижение цен на монохромные и даже цветные дисплеи до уровня, когда их можно купить за небольшую часть стоимости платы Arduino. И первое, что нужно решить при выборе графического дисплея, — это покупать монохромный или цветной дисплей. Это отчасти эстетическое решение, но на него могут повлиять такие факторы, как размер дисплея (в некоторых случаях большие цветные панели могут быть дешевле больших монохромных дисплеев), энергопотребление (дисплеи OLED, которые рассматриваются далее, более экономичны), сенсорная возможность (цветные дисплеи обычно обеспечивают в этом плане лучшую поддержку), а также тип подключения.

Графические дисплеи, которые пользуются популярностью в проектах Arduino, обычно основаны либо на жидкокристаллической (ЖК) технологии, либо на технологии органических светодиодов (OLED, Organic LED). Дисплеи на более продвинутой технологии OLED экономичнее обычных жидкокристаллических дисплеев, поскольку для них не требуется подсветка в условиях низкой освещенности. Но стоимость производства дисплеев OLED более высокая, и они также обычно имеют меньший размер, чем ЖК-дисплеи со сходной ценой. Цветные дисплеи OLED весьма дорогие, и те из них, что предлагаются для использования с Arduino, обычно очень маленького размера.

Обсуждение работы решения и возможных проблем

Исторически монохромные дисплеи были намного более дешевыми, чем цветные, но разрыв в цене быстро сокращается. Монохромные дисплеи рекомендуются к выбору, если вам нужен дисплей небольшого размера, или когда важно низкое энергопотребление. Кроме этого, скетчи для работы с монохромными дисплеями обычно меньшего размера, чем скетчи для цветных дисплеев. И этот момент следует учитывать в случае использования плат с ограниченным объемом доступной памяти.

Компания Adafruit предоставляет для OLED-дисплеев библиотеки SSD1306, SSD1325, SSD1331 и SSD1351. Полезной особенностью этих дисплеев является то, что их API подобен библиотекам компании Adafruit для интерфейса с цветными дисплеями. Это позволяет использовать в будущих проектах как монохромные, так и цветные дисплеи, или же просто заменить одноцветный дисплей цветным. Дополнительную информацию по дисплеям и библиотекам компании Adafruit вы найдете в соответствующей документации (**https://oreil.ly/RYQqV**).

Библиотека u8g2 для монохромных дисплеев реализует широкий круг возможностей и поддерживает свыше 60 разных контроллеров, включая контроллеры SSD1305, SSD1306, SSD1309, SSD1322, SSD1325, SSD1327, SSD1606, SH1106, T6963, RA8835, LC7981, PCD8544, PCF8812, UC1604, UC1608, UC1610, UC1611, UC1701, ST7565, ST7567, NT7534, IST3020, ST7920, LD7032 и KS0108, поэтому она однозначно рекомендуется к выбору. Полный список поддерживаемых библиотекой контроллеров ЖКД приведен на веб-сайте этой библиотеки (**https://oreil.ly/P_Hoh**).

Библиотека u8g2 поддерживает все стандартные интерфейсы, включая I^2C, SPI и параллельный, так что шансы, что библиотека u8g2 будет работать с любым используемым вами дисплеем, весьма велики. Исходный код библиотеки u8g2 можно загрузить здесь: **https://oreil.ly/3js9H**, а на ее веб-странице u8g2 wiki (**https://oreil.ly/D5UK4**) предлагается обширная документация.

Цветные дисплеи по определению предоставляют возможность отображать информацию разными цветами. Библиотека GFX компании Adafruit предоставляет стандартный набор графических примитивов, которые поддерживают многие библиотеки, предназначенные для работы с конкретным дисплеем, — например, библиотеки Adafruit_ST7735, Adafruit_HX8340B, Adafruit_HX8357D, Adafruit_ILI9340/1 и Adafruit_PCD8544. Библиотека GFX также пользуется популярностью благодаря обширной документации и большому количеству учебных пособий. В частности, в учебном пособии по адресу: **https://oreil.ly/3X4bh** рассматриваются все графические функции, характерные для всех графических библиотек компании Adafruit. Библиотека Adafruit Arcada (**https://oreil.ly/IHghh**) сочетает в себе поддержку GFX с большим набором функций, полезных для создания игр или богатого графического пользовательского интерфейса, включая различные типы пользовательского ввода.

Дополнительная информация

Библиотека Adafruit_TouchScreen (**https://oreil.ly/cP5Yc**) обеспечивает поддержку 4-проводных резистивных сенсорных экранов. Поддержку сенсорных экранов обеспечивает также библиотека Adafruit_FT6206 (**https://oreil.ly/JhdoN**).

Для владельцев недорогих сенсорных или простых TFT-экранов на основе контроллера дисплея ITDB02 (**https://oreil.ly/b5xcw**) или ILI9341 может представлять интерес библиотека UTFT (**https://oreil.ly/vcKn2**).

Библиотека URTouch (**https://oreil.ly/xVM9X**), сопутствующая библиотеке UTFT, обеспечивает поддержку сенсорных экранов.

Остерегайтесь дешевых дисплеев, не поддерживаемых или слабо поддерживаемых поставщиками

Из-за громадного количества предлагаемых на рынке графических дисплеев выбор конкретного дисплея может быть пугающей задачей. Если у вас нет знаний или опыта, которые позволили бы вам понимать высокотехническую информацию справочных листков контроллеров дисплея, не поддавайтесь искушению приобрести какой-либо сверхдешевый дисплей у поставщика, который не предоставляет техническую документацию и поддержку. В описании дисплея может указываться, что в комплект поставки входит программное обеспечение для платформы Arduino, но нередко случается так, что в действительности это обеспечение не работает с этим дисплеем или с последней версией среды Arduino IDE. Более безопасный подход — приобретать дисплеи у известных поставщиков, таких как, например, компания Adafruit или SparkFun, которые предоставляют обширную документацию, справочные пособия и форумы поддержки для своих продуктов.

11.10. Управление полноцветным жидкокристаллическим дисплеем

ЗАДАЧА

Требуется отображать полноцветную графику на графическом ЖКД на базе контроллера ST7735 или подобном.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 11.14. В скетче используются библиотека ST7735/ST7789 компании Adafruit, предоставляющая поддержку для цветных дисплеев TFT на базе контроллеров ST7735 и ST7789. Например, дисплея компании Adafruit с размером по диагонали 1,8 дюйма, смонтированного на адаптерной плате (артикул 358), или ее же дисплея IPS TFT с размером по диагонали 2 дюйма (артикул 4311). Вместо стандартного дисплея можно использовать дисплей HalloWing M0 компании Adafruit, в состав которого входит плата разработчика SAMD21, 8 Мбайт флеш-памяти, драйвер динамика и полноцветный ЖК-дисплей с размером по диагонали 1,44 дюйма, — все это смонтировано на печатной плате в форме черепа. Библиотека устанавливается с помощью Менеджера библиотек среды Arduino IDE (подробно установка библиотек описана в *главе 16*).

Скетч инициализирует дисплей, выводит три строки текста шрифтом разного размера, а затем начинает отображать анимацию желтого мячика, перемещающегося туда и обратно.

В случае использования дисплея на плате разработчика HalloWing (или любого другого подобного дисплея компании Adafruit) нужно будет добавить интернет-адрес (URL) менеджера плат Adafruit (**https://oreil.ly/t_F3z**) в среду Arduino IDE. После чего установить поддержку для плат с микроконтроллером SAMD компании Adafruit (**Инструменты | Плата | Менеджер плат**), а затем в меню **Инструменты | Плата** выбрать плату HalloWing M0.

Листинг 11.14. Скетч для работы с полноцветным ЖК-дисплеем на базе контроллера ST7735

```
/*
* Скетч Adafruit GFX ST7735
* Отображает на дисплее сначала текст, а затем перемещающийся мячик
*/

#include <Adafruit_GFX.h>    // Базовая графическая библиотека
#include <Adafruit_ST7735.h> // Графическая библиотека для дисплеев
                             // на базе контроллера ST7735s
#include <SPI.h>

// Определяем номера контактов платы Arduino для подключения дисплея.
// Они будут зависеть как от используемого дисплея, так и от
// платы Arduino
#define TFT_CS 39
#define TFT_RST 37
#define TFT_DC 38
#define TFT_BACKLIGHT 7
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

void setup(void)
{
   tft.initR(INITR_144GREENTAB); // Инициализируем контроллер ST7735
                                 // (в упаковке с зеленым ярлычком)

   pinMode(TFT_BACKLIGHT, OUTPUT);     // Контакт для подсветки
   digitalWrite(TFT_BACKLIGHT, HIGH); // Включаем подсветку

   tft.setRotation(2); // Это значение будет зависеть от способа установки дисплея
   tft.fillScreen(ST77XX_BLACK); // Заполняем экран черным цветом

   // Отображаем текст разными шрифтами
   tft.setCursor(0, 0);
   tft.setTextWrap(false);
   tft.setTextColor(ST77XX_RED);
```

```
    tft.setTextSize(1);
    tft.println("Small");
    tft.setTextColor(ST77XX_GREEN);

    tft.setTextSize(2);
    tft.println("Medium");

    tft.setTextColor(ST77XX_BLUE);
    tft.setTextSize(3);
    tft.println("Large");
}
int ballDir = 1; // Текущее направление движения
int ballDiameter = 8; // Диаметр
int ballX = ballDiameter; // Начальная позиция X

void loop()
{
    // Если мячик приближается к краю экрана, меняем направление
    // его движения на обратное
    if (ballX >= tft.width() - ballDiameter || ballX < ballDiameter)
    {
        ballDir *= -1;
    }

    ballX += ballDir; // Перемещаем позицию X мячика

    // Вычисляем позицию Y на основе местонахождения курсора
    int ballY = tft.getCursorY() + ballDiameter*2;

    // Желтый мячик
    tft.fillCircle(ballX, ballY, ballDiameter/2, 0xffff00);
    delay(25);
    // Стираем мячик
    tft.fillCircle(ballX, ballY, ballDiameter/2, 0x000000);
}
```

Обсуждение работы решения и возможных проблем

Для исполнения этого скетча не на плате разработчика с интегрированным дисплеем (т. е. не на плате HalloWing), а на отдельной плате Arduino (или совместимой с ней) нужно подключить к ней графический ЖК-дисплей, поддерживаемый библиотекой Adafruit ST7735/ST7789 для дисплеев, предлагаемых этой компанией. Схема подключения дисплея к плате Arduino Uno показана на рис. 11.7. Номера контактов для подключения линий MOSI и SCLK зависят от используемой платы (подключение других плат рассматривается в *разд. «Интерфейс SPI» главы 13*). Для платы Arduino Uno операторы `#define` скетча и строку инициализации дисплея нужно откорректировать следующим образом:

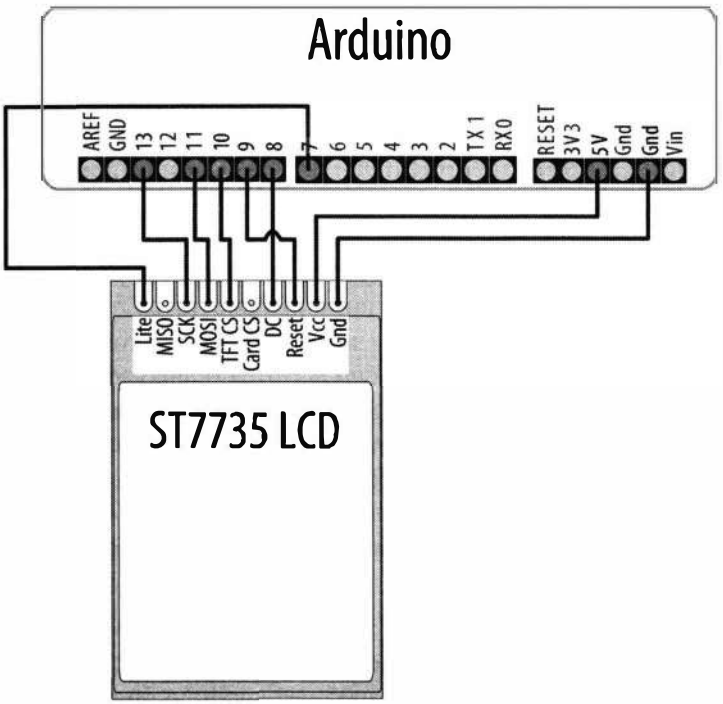


Рис. 11.7. Подключение дисплея LCD на базе контроллера ST7735 к плате Arduino

```
#define TFT_CS 10
#define TFT_RST 9
#define TFT_DC 8
#define TFT_BACKLIGHT 7
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);
```

Контакты линий выбора устройства и данных и команд определяются значениями констант `TFT_CS` и `TFT_DC`, которым можно присвоить другие номера контактов. Контакт, указанный в константе `TFT_RST`, служит для сброса дисплея, и ему также можно задать другой номер. При подключении дисплея к иным контактам платы Arduino, чем указано здесь, следует внести соответствующие корректировки в код скетча.

Сначала код в операторах `#define` присваивает идентификаторы номерам контактов, для которых отсутствуют идентификаторы, определенные программным обеспечением используемой платы, а затем инициализирует экземпляр класса библиотеки Adafruit_ST7735 как объект `tft`. В зависимости от используемой платы и номеров контактов платы Arduino, к которым подключен дисплей, код инициализации может быть иным. Например, можно использовать интерфейс SPI и задать другие номера контактов для линий MOSI и SCLK:

```
#define TFT_SCLK 5
#define TFT_MOSI 6
Adafruit_ST7735 tft =
   Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK, TFT_RST);
```

но это выльется в понижение производительности. Кроме того, если плата оснащена SD-картой, она не будет работать с программным обеспечением SPI.

В функции `setup()` скетч инициализирует дисплей, вызывая функцию `initR()`. Поскольку существует много вариантов этих дисплеев, обязательно изучите их свойства и характеристики по соответствующей документации (пример программы тестирования графики, входящей в состав библиотеки, содержит пояснительные комментарии для многих поддерживаемых плат). Например, есть группа вариантов дисплея, которые идентифицируются по цвету ярлыка, прикрепленного к защитной пленке экрана дисплея. Такие дисплеи инициализируются оператором, соответствующим цвету этого ярлыка. Например, для TFT-дисплея компании Adafruit с размером экрана по диагонали 1,8 дюйма используется оператор `INITR_BLACKTAB`. А дисплей с размером 1,44 дюйма, интегрированный в плату разработчика HalloWing M0, такой же, что и дисплей с размером 1,44 дюйма, имеющий зеленый ярлычок, однако установлен он «вверх ногами». Для этого дисплея можно использовать оператор инициализации `INITR_144GREENTAB`, а затем оператор `setRotation(2)` (портретный режим «вверх ногами») — как показано в скетче, или просто передать функции `initR()` параметру `INITR_HALLOWING`.

После инициализации скетч заполняет весь экран черным цветом, посредством функции `setCursor()` позиционирует курсор в требуемом месте экрана, а затем выводит на экран три строки текста — каждую следующую более крупным шрифтом, чем предыдущая. В функции `loop()` скетч перемещает мячик от одного края экрана к другому. Направление движения мячика задается значением переменной `ballDir()`: значение 1 означает перемещение вправо, а –1 — влево. Когда мячик попадает в позицию, когда он отстоит от стороны экрана на величину своей ширины, направление его движения меняется на обратное.

Дополнительная информация

Подробная информация на контроллер жидкокристаллического дисплея ST7735 приводится в его справочном листке: **https://oreil.ly/1S04i**.

11.11. Управление монохромным дисплеем OLED

ЗАДАЧА

Требуется отображать графику на монохромном дисплее OLED на базе контроллера SSD13*xx* или подобном.

РЕШЕНИЕ

В этом решении участвует поставляемый компанией Adafruit дисплей OLED на базе контроллера SSD1306 с интерфейсом SPI и размером матрицы экрана 128×32. В качестве интерфейса API используется библиотека Adafruit_SSD1306. Подключение дисплея к плате Arduino показано на рис. 11.8.

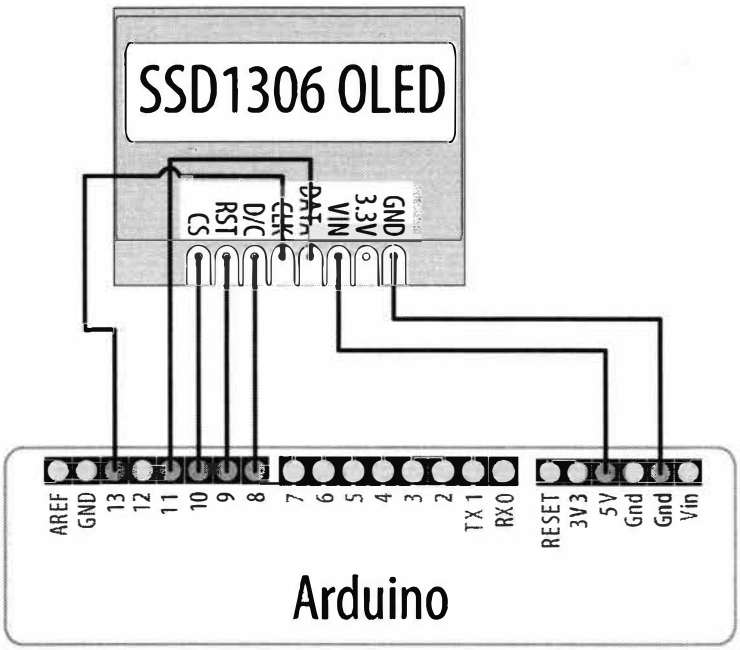


Рис. 11.8. Подключение дисплея OLED на базе контроллера SSD13*xx* к плате Arduino

Библиотека SSD1306 поддерживает широкий круг дисплеев OLED на базе указанного контроллера. Библиотека устанавливается с помощью Менеджера библиотек среды Arduino IDE (подробно установка библиотек описана в *главе 16*).

Скетч решения приводится в листинге 11.15. Здесь сначала выводится на экран прокручивающийся текст, а затем начинает перемещаться мячик от одной стороны экрана к другой.

Листинг 11.15. Скетч для работы с дисплеем OLED на базе контроллера SD13*xx*

```
/*
* Скетч OLED SSD13xx
* Сначала отображает на дисплее OLED текст, а затем начинает перемещать
  изображение мячика
*/

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

#define WIDTH 128
#define HEIGHT 32

#define OLED_DC 8
#define OLED_CS 10
```

```
#define OLED_RESET 9
Adafruit_SSD1306 display(WIDTH, HEIGHT, &SPI, OLED_DC, OLED_RESET, OLED_CS);

#define MODE SSD1306_SWITCHCAPVCC // Создаем напряжение питания дисплея
                                  // из питания 3,3 В в самом дисплее

void setup()
{
   Serial.begin(9600);
   if(!display.begin(MODE))
   {
      Serial.println("Could not initialize display");
                  // Не удалось инициализировать дисплей
      while(1); // halt
   }

   showAndScroll("Small", 1);
   showAndScroll("Medium", 2);
   showAndScroll("Large", 3);
}

// Show text and scroll it briefly
void showAndScroll(String text, int textSize)
{
   display.setTextColor(SSD1306_WHITE); // Отображаем текст белым цветом
   display.setCursor(0, 0); // Перемещаем курсор в позицию 0,0
   display.clearDisplay();  // Очищаем дисплей

   display.setTextSize(textSize);
   display.println(text);
   display.display();

   // Прокручиваем текст вправо в течение 3 секунд
   display.startscrollright(0x00, 0x0F);
   delay(3000);
   display.stopscroll();
}

int ballDir = 1;      // Текущее направление движения
int ballDiameter = 8;        // Диаметр мячика
int ballX = ballDiameter;    // Начальная позиция X
int ballY = ballDiameter*2;  // Начальная позиция Y

void loop()
{
   display.clearDisplay();
```

```
  // Если мячик приближается к краю экрана,
  // меняем направление его движения на обратное
  if (ballX >= WIDTH - ballDiameter || ballX < ballDiameter)
  {
    ballDir *= -1;
  }

  // Перемещаем позицию X мячика
  ballX += ballDir;
  // Draw a ball
  display.fillCircle(ballX, ballY, ballDiameter/2, SSD1306_INVERSE);
  display.display();
  delay(25);

  // Стираем мячик
  display.fillCircle(ballX, ballY, ballDiameter/2, SSD1306_INVERSE);
  display.display();
}
```

Обсуждение работы решения и возможных проблем

Контакты линий выбора устройства и данных и команд определяются в скетче значениями идентификаторов `TFT_CS` и `TFT_DC`. Контакт `OLED_RESET` служит для сброса дисплея. Вместо этих номеров контактов можно использовать любые другие. Номера контактов для подключения линий MOSI и SCLK зависят от используемой платы. На рис. 11.8 показано подключение платы Arduino Uno. Подключение других плат рассматривается в *разд. «Интерфейс SPI» главы 13*.

Это решение ориентировано на аппаратный интерфейс SPI. Чтобы задействовать программный интерфейс SPI, для каждого из его контактов нужно добавить оператор `#define`, подключить эти контакты и применить другой конструктор:

```
#define OLED_CLK 5
#define OLED_MOSI 6
Adafruit_SSD1306 display(WIDTH, HEIGHT,
                         OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);
```

Библиотека Adafruit поддерживает широкий круг дисплеев на базе контроллера SSD1306, поэтому в случае использования другого дисплея код скетча нужно соответственно откорректировать. Если дисплей имеет другой размер, но также основан на интерфейсе SPI, вам, скорее всего, придется просто откорректировать операторы `#define` для идентификаторов `WIDTH` и `HEIGHT`. Если вместо интерфейса SPI дисплей использует интерфейс I^2C, для его подключения потребуется меньшее количество линий. Контакт сброса надо будет подключить так же, как и для интерфейса SPI, но для обмена данными и командами между дисплеем и платой Arduino останется подключить только линии SCL и SDA. Также нужно будет откорректировать скетч, подключив заголовочный файл *Wire.h* и используя другой вариант инициализации:

```
#include <Wire.h>
#define WIDTH 128
#define HEIGHT 32
#define OLED_RESET 13
Adafruit_SSD1306 display(WIDTH, HEIGHT, &Wire, OLED_RESET);
```

В результате инициализации дисплея создается экземпляр объекта библиотеки Adafruit_SSD1306, методы (функции) которого используются для управления дисплеем. В функции `setup()` скетч запускает дисплей в режиме работы `SSD1306_SWITCHCAPVCC` (чтобы дисплей использовал свои ресурсы для создания напряжения питания из напряжения 3,3 В). Затем с помощью функции `showAndScroll()` скетч отображает прокручивающийся текст шрифтами трех разных размеров. Эта функция конфигурирует дисплей для отображения текста белого цвета, очищает экран и устанавливает курсор в верхнем левом углу экрана. Затем задается размер шрифта, выполняется команда вывода текста на экран, после чего вызывается функция `display.dispaly()` чтобы собственно отобразить текст на экране. Этот подход отличается от подхода, используемого с цветным ЖК-дисплеем в *разд. 11.10*, где все, что отправлялось на экран, немедленно на нем и отображалось. Отображенный текст прокручивается в течение трех секунд.

В главном цикле `loop()` скетч прорисовывает на экране мячик и перемещает его туда и обратно между сторонами экрана. При каждой прорисовке мячика скетч изменяет его цвет с белого на черный и обратно, используя для этого оператор `SSD1306_INVERSE`. При каждой итерации цикла `loop()` скетч инкрементирует значение переменной `ballX`, пока мячик не достигнет правой стороны экрана, после чего это значение начинает декрементироваться до тех пор, пока мячик не достигнет левой стороны. Затем скетч на короткое время отображает мячик, после чего стирает его.

Подобную графику можно создать, используя библиотеку u8g2. Код будет немного другой, но по большей части такой же. Самое большое различие состоит в том, что вместо вывода графики на экран с последующим вызовом функции `display.display()` для ее отображения будет использоваться страничный буфер библиотеки u8g2 для поэтапной прорисовки графики на дисплее. Для этого сначала нужно вызвать функцию `u8g2.firstPage()`, а затем запустить цикл `do...while`, в конце которого вызывается функция `u8g2.nextPage()`. Команды вывода на экран помещаются внутри цикла, и буфер u8g2 обрабатывается в цикле столько раз, сколько нужно для отображения всего экрана. Библиотека u8g2 поддерживает большой круг монохромных дисплеев и вариантов для каждого типа дисплея — например, с аппаратным интерфейсом SPI, программным интерфейсом SPI, интерфейсом I2C, а также комбинациями этих интерфейсов для разных размеров буфера кадра/страницы. Полный список поддерживаемых дисплеев и соответствующие функции их инициализации приводятся на веб-странице библиотеки (**https://oreil.ly/XhGA0**).

В листинге 11.16 приводится скетч, использующий библиотеку U8g2 для вывода на одноцветный дисплей OLED графики, как в предыдущих скетчах.

Листинг 11.16. Использование библиотеки U8g2 для работы с одноцветным OLED-дисплеем

```
/*
* Скетч u8g2 oled
* Отображает разный текст, затем перемещает мячик по экрану
*/

#include <Arduino.h>
#include <U8g2lib.h>
#include <SPI.h>

#define OLED_DC 8
#define OLED_CS 10
#define OLED_RESET 9
U8G2_SSD1306_128X32_UNIVISION_2_4W_HW_SPI u8g2(U8G2_R0, OLED_CS,
OLED_DC, OLED_RESET);

u8g2_uint_t displayWidth;

void setup(void)
{
   u8g2.begin();
   u8g2.setFontPosTop();
   displayWidth = u8g2.getDisplayWidth();

   showAndScroll("Small", u8g2_font_6x10_tf);
   showAndScroll("Medium", u8g2_font_9x15_tf);
   showAndScroll("Large", u8g2_font_10x20_tf);
}

int ballDir = 1;           // Текущее направление движения
int ballRadius = 4;        // Радиус
int ballX = ballRadius*2;  // Начальная позиция X
int ballY = ballRadius*4;  // Начальная позиция Y

void loop(void)
{
   u8g2.firstPage(); // Цикл прорисовки изображения
   do
   {
      // Если мячик приближается к краю экрана,
      // меняем направление его движения на обратное
      if (ballX >= displayWidth - ballRadius*2 || ballX < ballRadius*2)
      {
         ballDir *= -1;
      }
```

```
      ballX += ballDir; // Перемещаем позицию X мячика
      u8g2.drawDisc(ballX, ballY, ballRadius); // Прорисовываем мячик
   }
   while ( u8g2.nextPage() );
   delay(25);
}

void showAndScroll(String text, uint8_t *font)
{
   for (int i = 0; i < 20; i++)
   {
      u8g2.firstPage(); // Цикл прорисовки изображения
      do
      {
         u8g2.setFont(font);
         u8g2.drawStr(10 + i, 10, text.c_str());
      }
      while ( u8g2.nextPage() );
      delay(125);
   }
}
```

Дополнительная информация

Дополнительная информация по дисплеям OLED компании Adafruit приведена в соответствующей документации (**https://oreil.ly/Bn3Y8**).

Подробная информация по микросхеме контроллера дисплея SSD1306 приведена в ее справочном листке (**https://oreil.ly/TBjHf**).

ГЛАВА 12

Работа с временем и датами

12.0. Введение

Управление временем является одним из фундаментальных элементов интерактивных вычислений. В этой главе мы рассмотрим встроенные временны́е функции Arduino и познакомимся со многими дополнительными методами обработки временны́х задержек, измерения периодов времени и учета реального времени и дат. Мы научимся использовать встроенную функцию Arduino для реализации пауз в скетчах и разберемся с продвинутыми способами выполнения периодических операций. Мы также узнаем, как измерять течение времени и как использовать внешние часы реального времени для отслеживания времени и дат.

12.1. Использование функции *millis()* для определения длительности периода времени

ЗАДАЧА

Требуется знать, сколько прошло времени после возникновения определенного события. Например, сколько времени кнопка удерживается нажатой.

РЕШЕНИЕ

Скетч в листинге 12.1 использует функцию `millis()`, чтобы определить длительность нажатия кнопки, и выводит это значение в окно монитора порта (подробно о подключении кнопки рассказано в *разд. 5.2*).

Листинг 12.1. Использование функции `millis()` для определения истекшего времени

```
/*
Скетч millisDuration
Возвращает время (в миллисекундах), в течение которого кнопка удерживается нажатой
*/

const int switchPin = 2; // Номер контакта кнопки (ввод)

unsigned long startTime; // Значение millis() при нажатии кнопки
unsigned long duration;  // Переменная для хранения значения длительности
```

```
void setup()
{
   pinMode(switchPin, INPUT_PULLUP);
   Serial.begin(9600);
}

void loop()
{
   if(digitalRead(switchPin) == LOW)
   {
      // Переходим сюда, если нажата кнопка
      startTime = millis();
      while(digitalRead(switchPin) == LOW); // Ожидаем здесь
            // в течение времени удерживания кнопки нажатой
      unsigned long duration = millis() - startTime;
      Serial.println(duration);
   }
}
```

Обсуждение работы решения и возможных проблем

Функция `millis()` возвращает время (в миллисекундах), истекшее с начала исполнения текущего скетча.

Возвращаемое функцией `millis()` значение переполнится (счет истекшего времени начнет заново вестись с нуля) после приблизительно 50 дней. В *разд. 12.4* и *12.5* рассказано, как использовать библиотеку TimeLib для работы с интервалами времени в диапазоне от секунд до нескольких лет.

Сохранив время начала события, можно определить его длительность, отняв время начала от текущего времени, как показано в следующей строке кода:

```
unsigned long duration = millis() - startTime;
```

Дополнительная информация

Более подробная информация по функции `millis()` приводится на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/WwCZl**).

В *разд. 12.4* и *12.5* рассказано, как использовать библиотеку TimeLib для работы с интервалами времени в диапазоне от секунд до нескольких лет.

12.2. Создание пауз в скетче

ЗАДАЧА

Требуется, чтобы скетч выдержал паузу в течение определенного периода времени. Длительность паузы может задаваться определенным количеством миллисекунд или же в секундах, минутах, часах или днях.

РЕШЕНИЕ

Во многих скетчах этой книги для выдерживания паузы используется функция `delay()`. Эта функция приостанавливает исполнение кода скетча на количество миллисекунд, передаваемых ей в качестве параметра (одна секунда содержит тысячу миллисекунд). В скетче из листинга 12.2 показано, как использовать функцию `delay()`, чтобы получать паузы практически любой длительности.

Листинг 12.2. Создание пауз в исполнении кода с помощью функции `delay()`

```
/*
* Скетч delay
*/

const unsigned long oneSecond = 1000; // Одна секунда содержит
                                      // 1000 миллисекунд
const unsigned long oneMinute = oneSecond * 60;
const unsigned long oneHour = oneMinute * 60;
const unsigned long oneDay = oneHour * 24;

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Needed on Leonardo and ARM-based boards
}

void loop()
{
   Serial.println("delay for 1 millisecond"); // Пауза в 1 миллисекунду
   delay(1);
   Serial.println("delay for 1 second"); // Пауза в 1 секунду
   delay(oneSecond);
   Serial.println("delay for 1 minute"); // Пауза в 1 минуту
   delay(oneMinute);
   Serial.println("delay for 1 hour");   // Пауза в 1 час
   delay(oneHour);
   Serial.println("delay for 1 day");    // Пауза в 1 день
   delay(oneDay);
   Serial.println("Ready to start over"); // Можно начинать снова
}
```

Обсуждение работы решения и возможных проблем

Поскольку функция `delay()` ограничена максимальным значением целого числа, она имеет диапазон значений от одной тысячной секунды до 25 дней, если передаваемый ею параметр имеет тип `long`. При использовании параметра типа `unsigned long`

максимальное значение удваивается до почти 50 дней (типы данных подробно рассматриваются в *главе 2*).

Для более коротких пауз можно использовать функцию `delayMicroseconds()`. Одна миллисекунда содержит 1000 микросекунд, а одна секунда — 1 миллион. Функция `delayMicroseconds()` может создавать паузы длительностью от одной микросекунды до 16 миллисекунд (но для пауз длительностью больше чем несколько тысяч микросекунд следует использовать функцию `delay()`):

```
delayMicroseconds(10); // Пауза длительностью в 10 микросекунд
```

Функции `delay()` и `delayMicroseconds()` создают паузы длительностью минимум в период времени, указанный в параметре, но пауза может длиться чуть дольше, если в течение ее происходят прерывания.

Недостаток использования функции `delay()` состоит в том, что в течение создаваемой ею паузы скетч не может ничего делать. В примере BlinkWithoutDelay (**Файл | Примеры | 02. Digital | BlinkWithoutDelay**) демонстрируется другой подход к созданию пауз в скетче. Он состоит в сохранении в переменной `previousMillis` времени последнего выполнения действия с последующей периодической проверкой значения функции `millis()` (которое определяется внутренними часами, отсчитывающими каждую миллисекунду в течение исполнения скетча). Когда разница между текущим значением функции `millis()` и значением переменной `previousMillis` становится равной или большей, чем заданный интервал времени, скетч выполняет требуемое действие — например, мигает светодиодом. В листинге 12.3 приводится сокращенная версия этого скетча.

Листинг 12.3. Использование переменной `previousMillis` для реализации паузы

```
int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 1000;

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

void loop()
{
   unsigned long currentMillis = millis();
   if (currentMillis - previousMillis >= interval)
   {
      previousMillis = currentMillis;
      if (ledState == LOW)
      {
         ledState = HIGH;
      }
```

```
      else
      {
         ledState = LOW;
      }
      digitalWrite(LED_BUILTIN, ledState);
   }
   // Здесь можно разместить код для выполнения других действий
}
```

В листинге 12.4 демонстрируется способ организации этой логики с помощью функции `myDelay()`, которая создает паузу в главном цикле `loop()`, но позволяет в течение этой паузы выполнять другие действия. Эту функциональность можно использовать требуемым образом, но в рассматриваемом примере она задействована для включения или выключения светодиода каждые 250 миллисекунд.

Листинг 12.4. Использование функции `myDelay()`

```
/*
* Скетч, использующий функцию myDelay() для мигания светодиодом
  с заданной частотой
*/

const int ledPin = LED_BUILTIN; // Номер контакта для подключения светодиода

int ledState = LOW; // Переменная для установки состояния светодиода
unsigned long previousMillis = 0; // Переменная для хранения
                // последнего времени обновления светодиода

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

void loop()
{
   if (myDelay(blink, 250))
   {
      Serial.println(millis() / 1000.0); // Выводим в окно
               // монитора порта истекшее время в секундах
   }
}

/*
* Выполняем указанную функцию, возвращаем true, если выполнили
*/
bool myDelay(void (*func)(void), long interval)
```

```
{
   unsigned long currentMillis = millis();
   if (currentMillis - previousMillis >= interval)
   {
      // Сохраняем последнее время изменения состояния светодиода
      previousMillis = currentMillis;
      func(); // Вызываем функцию
      return true;
   }
   return false;
}

void blink()
{
   // Если светодиод выключен, включаем его, и наоборот
   if (ledState == LOW)
   {
      ledState = HIGH;
   }
   else
   {
      ledState = LOW;
   }
   digitalWrite(ledPin, ledState);
}
```

Параметр `(*func)(void)` в определении функции `myDelay()` означает, что аргумент `func` представляет собой указатель на функцию типа `void`, которая не принимает никаких аргументов (`(void)`). Таким образом, при каждом вызове функции `func()` в функции `myDelay()` в действительности вызывается функция `blink()`. При достижении заданного значения временно́го интервала функция `myDelay()` сбрасывает значение переменной `previousMillis`, вызывает функцию `blink` и возвращает значение `true`.

Но при программировании в стиле Arduino рекомендуется избегать применения в скетчах указателей, резервируя их для использования в библиотеках, чтобы не сбивать с толку начинающих программистов, которые могут воспользоваться вашим скетчем. Поэтому другим подходом будет задействовать какую-либо библиотеку стороннего разработчика, доступную для установки с помощью Менеджера библиотек, — например, библиотеку Tasker (**https://oreil.ly/04Yz6**). В листинге 12.5 приводится скетч с использованием этой библиотеки. Скетч мигает с разной частотой двумя разными светодиодами: встроенным светодиодом платы и светодиодом, подключенным к контакту 10. В этом скетче мы избегаем необходимости сохранения состояния каждого контакта в отдельной переменной благодаря использованию функции `digitalRead()` для определения текущего состояния светодиода.

Листинг 12.5. Пример использования библиотеки Tasker

```
/*
* Скетч Tasker demo
*/

#define TASKER_MAX_TASKS 2 // Количество требуемых задач

#include <Tasker.h>

// Объявляем экземпляр объекта Tasker
Tasker tasker;

void setup()
{
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(10, OUTPUT);
   // Мигаем встроенным светодиодом раз в секунду
   tasker.setInterval(blink, 1000, LED_BUILTIN);
   // Мигаем светодиодом на контакте 10 два раза в секунду
   tasker.setInterval(blink, 500, 10);
}

void loop()
{
   tasker.loop(); // Выполняем ожидающие обработки задания
}

void blink(int pinNumber)
{
   bool ledState = !digitalRead(pinNumber); // Переключаем текущее
                                            // состояние контакта на обратное
   if (ledState)
   {
      digitalWrite(pinNumber, HIGH);
   }
   else
   {
      digitalWrite(pinNumber, LOW);
   }
}
```

Дополнительная информация

Более подробная информация по функции `delay()` приводится на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/6_Jnj**).

12.3. Точное измерение длительности импульса

ЗАДАЧА

Требуется определить длительность импульса с точностью до микросекунд. Например, точно измерить длительность *импульса* высокого (HIGH) или низкого (LOW) уровня на контакте. Импульс — это переход цифрового сигнала с низкого уровня на высокий и обратно на низкий, или с высокого на низкий и обратно на высокий.

РЕШЕНИЕ

Эта задача решается скетчем, приведенным в листинге 12.6. В нем используется функция `pulseIn()`, которая возвращает длительность в микросекундах импульса на указанном цифровом контакте платы Arduino. В частности, скетч измеряет длительность импульсов высокого и низкого уровней, создаваемых функцией `analogWrite()` (эта функция подробно рассматривается в *разд. «Аналоговый вывод» главы 7*) и выводит полученные результаты в окно монитора порта. Поскольку импульсы функции `analogWrite()` создаются самой платой Arduino, никакого внешнего оборудования подключать не требуется.

Листинг 12.6. Измерение длительности импульса

```
/*
Скетч PulseIn
Измеряет длительность импульсов высокого и низкого уровней,
создаваемых функцией analogWrite()
*/

const int inputPin = 3; // Выходной аналоговый контакт,
   // длительность сигнала на котором будет измеряться

unsigned long val; // Переменная для хранение значения,
                   // возвращенного функцией pulseIn

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

void loop()
{
   analogWrite(inputPin, 128);
   Serial.print("Writing 128 to pin "); // Выводим на контакт значение 128
   Serial.print(inputPin);
   printPulseWidth(inputPin);
   analogWrite(inputPin, 254);
```

```
    Serial.print("Writing 254 to pin "); // Выводим на контакт значение 254
    Serial.print(inputPin);
    printPulseWidth(inputPin);
    delay(3000);
}

void printPulseWidth(int pin)
{
    val = pulseIn(pin, HIGH);
    Serial.print(": High Pulse width = "); // Длительность импульса высокого уровня
    Serial.print(val);
    val = pulseIn(pin, LOW);
    Serial.print(", Low Pulse width = "); // Длительность импульса низкого уровня
    Serial.println(val);
}
```

Обсуждение работы решения и возможных проблем

В окне монитора порта должен отобразиться следующий текст:

```
Writing 128 to pin 3: High Pulse width = 989, Low Pulse width = 997
Writing 254 to pin 3: High Pulse width = 1977, Low Pulse width = 8
```

Функция `pulseIn()` может измерить длительность импульсов как высокого, так и низкого уровня:

```
pulseIn(pin, HIGH); // возвращает длительность в микросекундах
                    // импульса высокого уровня
pulseIn(pin, LOW);  // возвращает длительность в микросекундах
                    // импульса низкого уровня
```

Функция `pulseIn()` ожидает начала импульса до истечения времени тайм-аута. По умолчанию длительность тайм-аута составляет одну секунду, но это время можно изменить, указывая требуемое время тайм-аута в третьем параметре функции (обратите внимание на то, что 1000 микросекунд составляют 1 миллисекунду):

```
pulseIn(pin, HIGH, 5000); // Время ожидания начала импульса равно 5 миллисекунд
```

Время тайм-аута имеет значение только в том случае, если импульс не начнется в течение заданного периода. Когда же функция обнаруживает начало импульса, она начинает измерять его длительность и не возвращается до тех пор, пока не закончится импульс.

Функция `pulseIn()` может измерять импульсы длительностью от приблизительно 10 микросекунд до трех минут, но замеры более длинных импульсов могут оказаться не очень точными.

Дополнительная информация

Более подробная информация по функции `pulseIn()` приводится на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/Qnvoy**).

В *разд. 6.5* демонстрируется использование функции `pulseIn()` для измерения длительности импульса, выдаваемого ультразвуковым датчиком расстояния.

Дополнительная информация по работе с аппаратными прерываниями приведена в *разд. 18.2*.

12.4. Использование платы Arduino в качестве часов

ЗАДАЧА

Требуется использовать в скетче текущее время дня (часы, минуты и секунды), не подключая к плате Arduino дополнительное оборудование для измерения времени.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 12.7, в котором используется библиотека TimeLib для определения и отображения текущего времени. Библиотека TimeLib устанавливается стандартным способом с помощью Менеджера библиотек среды Arduino IDE. Найти библиотеку можно по ключевому слову `timekeeping`.

Листинг 12.7. Использование библиотеки TimeLib для определения текущего времени

```
/*
* Скетч Time
*/

#include <TimeLib.h>

void setup()
{
   Serial.begin(9600);
   setTime(12,0,0,1,1,2020); // Устанавливаем время на 12:00 1 января 2020 г.
}

void loop()
{
   digitalClockDisplay();
   delay(1000);
}

// Дополняем числа ведущим )
String padDigits(int digit)
{
   String str = String("0") + digit;       // Вставляем 0 перед цифрой
   return str.substring(str.length() - 2); // Удаляем все символы,
                                           // кроме двух последних
}
```

```
void digitalClockDisplay()
{
   String timestr = String(hour()) + ":" + padDigits(minute()) +
                    ":" + padDigits(second());
   Serial.println(timestr);
   String datestr = String(year()) + "-" + padDigits(month()) +
                    "-" + padDigits(day());
   Serial.println(datestr);
}
```

Обсуждение работы решения и возможных проблем

Библиотека TimeLib позволят скетчу Arduino отслеживать текущие даты и время.

Многие платы Arduino используют кварцевый кристалл для измерения времени с точностью до пары секунд в день, но у них нет батареи, чтобы запоминать время, когда питание платы отключено. Вследствие этого при каждом запуске скетча отсчет времени начнет вестись с нуля, поэтому нам нужно задать какое-либо исходное время, используя для этого функцию `setTime()`. В нашем скетче время устанавливается на 12:00 дня 1 января 2020 года.

Библиотека TimeLib использует стандарт времени UNIX, которое также называется временем POSIX или временем Epoch. Значения в этом стандарте представляют количество секунд, истекших с 1 января 1970 года. Опытные программисты на языке C могут увидеть, что это то же самое время, что и время `time_t`, используемое в библиотеке C стандарта ISO для хранения значений времени.

Конечно же, гораздо более полезно установить текущее локальное время, а не какое-то произвольное постоянное его значение. Скетч в листинге 12.8 получает числовое значение времени (количество секунд, истекших с 1 января 1970 г.) через последовательный интерфейс и на основе этого значения устанавливает локальное время. Текущее значение времени UNIX можно ввести посредством монитора порта. Это значение можно узнать в том числе и на веб-сайте EpochConverter **(https://www.epochconverter.com)**.

Листинг 12.8. Установка локального времени по значению текущего времени UNIX

```
/*
* Скетч SetTimeSerial
* Устанавливает время на основе значения, полученного через
  последовательный интерфейс. Это упрощенная версия скетча
  примера TimeSerial из библиотеки TimeLib.
*
* Устанавливаем время, отправляя скетчу через монитор порта строку,
  начинающуюся буквой T, за которой следует 10 цифр, обозначающих
  количество секунд, истекших с 1 января 1970 г.
  Например, строка T1569888000 представляет 12:00 дня 1 октября 2019 г.
*/
```

```
#include <TimeLib.h>

#define TIME_HEADER 'T' // Ярлык заголовка для сообщения синхронизации
            // времени, передаваемого по последовательному интерфейсу

void setup()
{
   Serial.begin(9600);
   Serial.println("Waiting for time sync message");
               // Ожидаем сообщение о синхронизации времени,
               // передаваемого по последовательному интерфейсу
}

void loop()
{
   if(Serial.available())
   {
      processSyncMessage();
   }
   if(timeStatus() != timeNotSet)
   {
      // Отображаем время и дату
      digitalClockDisplay();
   }
   delay(1000);
}

// Дополняем числа ведущим 0
String padDigits(int digit)
{
   String str = String("0") + digit;       // Вставляем 0 перед цифрой
   return str.substring(str.length() - 2); // Удаляем все символы,
                                           // кроме двух последних
}

void digitalClockDisplay()
{
   String timestr = String(hour()) + ":" + padDigits(minute()) +
                                 ":" + padDigits(second());
   Serial.println(timestr);
   String datestr = String(year()) + "-" + padDigits(month()) +
                                 "-" + padDigits(day());
   Serial.println(datestr);
}

// Выполняем парсинг сообщения синхронизации
void processSyncMessage()
```

```
{
   time_t pctime = 0;
   if(Serial.find(TIME_HEADER))
   {
      pctime = Serial.parseInt();
      setTime(pctime); // Устанавливаем часы на время, полученное
                       // через последовательный интерфейс
   }
}
```

Код для отображения времени и даты такой же, как и в скетче из листинга 12.7, но теперь скетч ждет получения значения времени через последовательный интерфейс (обмен числовыми данными через последовательный интерфейс подробно рассматривается в *разд. 4.3*).

Библиотека TimeLib содержит в примерах скетч Processing SyncArduinoClock (находится в папке Time/Examples/Processing/SyncArduinoClock). По щелчку мыши этот скетч отправляет плате Arduino текущее время компьютера. Запустите этот скетч на исполнение, убедившись, что последовательный порт подключен к плате Arduino (исполнение скетчей Processing, взаимодействующих с платой Arduino, рассматривается в *главе 4*). В консоли окна Processing (черная область внизу этого окна) должно отобразиться сообщение **Waiting for time sync message** (Ожидаем сообщение синхронизации времени), отправленное скетчем Arduino. Щелкните мышью в окне приложения Processing (в сером квадрате размером 200×200 пикселов) — в консоли окна Processing должно отобразиться время, переданное скетчем Arduino.

Установить часы на плате Arduino также можно, передав скетчу Arduino через монитор порта текущее время UNIX, которое можно получить на одном из многих веб-сайтов, предоставляющих эту услугу, — например, на веб-сайте Epoch Converter (**https://www.epochconverter.com**). Предоставляемое таким веб-сайтом значение должно быть в микросекундах (10-разрядное значение — по крайней мере, до 2286 года). Значение в миллисекундах будет в 1000 раз большим. Скопируйте 10-разрядное число текущего времени UNIX, вставьте его в строку исходящих сообщений монитора порта. Введите перед числом английскую букву `T` и нажмите кнопку **Отправить**. Если отправить таким образом, например, это сообщение:

```
T1282041639
```

скетч Arduino должен ответить, отображая каждую секунду текущее время в окне входящих сообщений монитора порта:

```
10:40:49 17 8 2019
10:40:50 17 8 2019
10:40:51 17 8 2019
10:40:52 17 8 2019
10:40:53 17 8 2019
10:40:54 17 8 2019
. . .
```

Время также можно устанавливать, используя кнопки или другие устройства ввода — такие как, например, датчик наклона, джойстик или поворотный энкодер.

Скетч в листинге 12.9 использует две кнопки для смещения значения текущего времени вперед и назад. В нем применены функции `digitalClockDisplay()` и `printDigits()` из скетча решения *разд. 12.3*, поэтому скопируйте определения этих функций, прежде чем загружать и исполнять этот скетч.

Подключение кнопок для использования в скетче показано на рис. 12.1 (подробно о подключении кнопок и работе с ними рассказано в *разд. 5.2*).

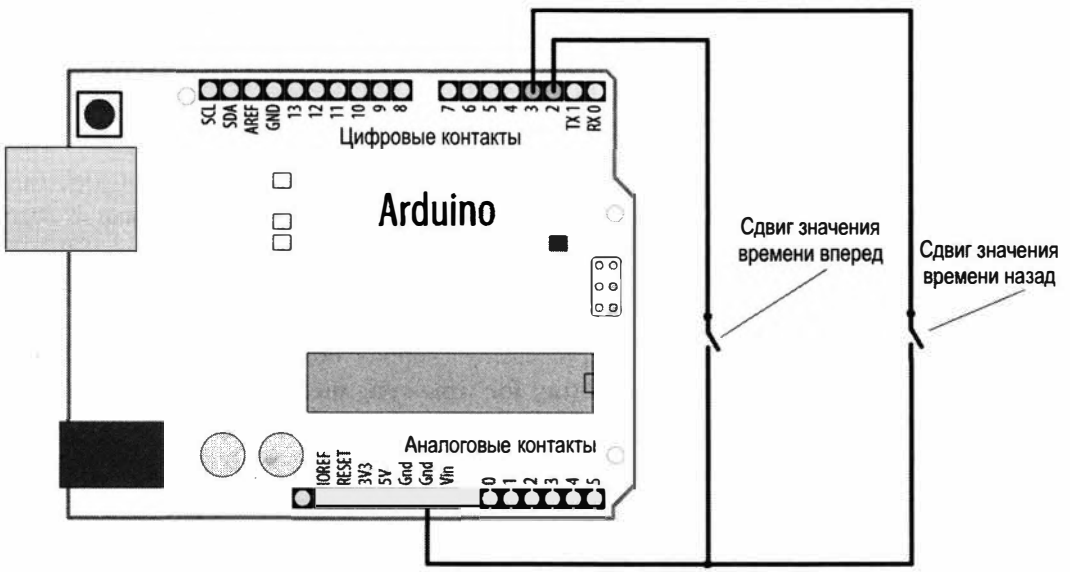


Рис. 12.1. Подключение кнопок для установки значения времени

Листинг 12.9. Скетч для установки значения времени с помощью двух кнопок

```
/*
Скетч AdjustClockTime
Кнопки на контактах 2 и 3 устанавливают значение времени
*/

#include <TimeLib.h>

const int btnForward = 2; // Кнопка для увеличения значения времени
const int btnBack = 3;    // Кнопка для уменьшения значения времени

unsigned long prevtime; // Время последнего отображения значения времени

void setup()
{
   pinMode(btnForward, INPUT_PULLUP); // Подключаем внутренние повышающие резисторы
   pinMode(btnBack, INPUT_PULLUP);
```

```
   setTime(12,0,0,1,1,2020); // Устанавливаем начальное время
                             // на 12:00 1 января 2020 г.
   Serial.begin(9600);
}

void loop()
{
   prevtime = now(); // Отмечаем время
   while( prevtime == now() ) // Исполняем этот цикл, пока не истечет секунда
   {
      // Проверяем, нажата ли кнопка, пока ожидаем истечения секунды
      if(checkSetTime())
      prevtime = now(); // Время изменилось, поэтому сбрасываем время начала
   }
   digitalClockDisplay();
}

// Функция проверяет, нужно ли корректировать значение времени
// Возвращает true, если время изменилось
bool checkSetTime()
{
   int step; // количество секунд корректировки назад,
             // если значение отрицательное
   bool isTimeAdjusted = false; // Если была выполнена корректировка
                                // значения времени, задаем значение true
   step = 1; // Можно увеличить значение

   while(digitalRead(btnForward)== LOW)
   {
      adjustTime(step);
      isTimeAdjusted = true; // Для извещения пользователя об изменении
                             // значения времени
      step = step + 1;       // Следующий шаг корректировки будет большим
      digitalClockDisplay(); // Обновляем отображаемое значение времени
      delay(100);
   }
   step = -1; // Для отрицательных чисел корректировка выполняется
              // в обратную сторону
   while(digitalRead(btnBack)== LOW)
   {
      adjustTime(step);
      isTimeAdjusted = true; // Для извещения пользователя об изменении
                             // значения времени
      step = step - 1; // Следующий шаг корректировки будет большим
                       // отрицательным числом
      digitalClockDisplay(); // Обновляем отображаемое значение времени
      delay(100);
   }
```

```
   return isTimeAdjusted; // Если была выполнена корректировка
                          // значения времени, извещаем пользователя
}

// Дополняем числа ведущим 0
String padDigits(int digit)
{
   String str = String("0") + digit; // Вставляем 0 перед цифрой
   return str.substring(str.length() - 2); // Удаляем все символы,
                                           // кроме двух последних
}

void digitalClockDisplay()
{
   String timestr = String(hour()) + ":" + padDigits(minute()) +
                                     ":" + padDigits(second());
   Serial.println(timestr);
   String datestr = String(year()) + "-" + padDigits(month()) +
                                     "-" + padDigits(day());
   Serial.println(datestr);
}
```

В листинге 12.10 приводится вариант предыдущего скетча, в котором направление и скорость изменения значения текущего времени определяются положением потенциометра. В этом скетче используются функции `digitalClockDisplay()` и `printDigits()` из скетча решения *разд. 12.3*, поэтому скопируйте определения этих функций, прежде чем загружать и исполнять этот скетч. Подключение потенциометра и кнопки показано на рис. 12.2. В случае использования платы с рабочим напряжением 3,3 В, которая не может работать с логическими уровнями с величиной 5 В, положительный вывод потенциометра нужно подключить к контакту 3V3 платы Arduino, а не к контакту 5V.

Листинг 12.10. Определение направления и скорости изменения значения текущего времени с помощью потенциометра

```
#include <TimeLib.h>

const int potPin = A0; // Номер контакта для подключения  потенциометра,
                       // управляющего направлением и скоростью изменения
                       // значения текущего времени
const int buttonPin = 2; // Номер контакта для подключения кнопки
                         // корректировки значения текущего времени

unsigned long prevtime; // Время последнего отображения значения времени

void setup()
{
```

```
   digitalWrite(buttonPin, HIGH); // Подключаем внутренние повышающие резисторы
   setTime(12,0,0,1,1,2020);      // Устанавливаем начальное время
                                  // на 12:00 1 января 2020 г.
   Serial.begin(9600);
}

void loop()
{
   prevtime = now();          // Отмечаем время
   while( prevtime == now() ) // Исполняем этот цикл, пока не истечет секунда
   {
      // Проверяем, нажата ли кнопка, пока ожидаем истечения секунды
      if(checkSetTime())
      prevtime = now(); // Время изменилось, поэтому сбрасываем время начала
   }
   digitalClockDisplay();
}

// Функция проверяет, нужно ли корректировать значение времени
// Возвращает true, если время изменилось
bool checkSetTime()
{
   int value; // Считываемое с потенциометра значение
   int step;  // Количество секунд корректировки (назад,
              // если значение отрицательное)
   bool isTimeAdjusted = false; // Если была выполнена корректировка
                          // значения времени, задаем значение true
   while(digitalRead(buttonPin)== LOW)
   {
      // Переходим сюда, пока нажата кнопка
      value = analogRead(potPin);   // Считываем значение выходного
                                    // сигнала потенциометра
      step = map(value, 0,1023, 10, -10); // Масштабируем полученное
                        // значение к требуемому диапазону значений
      if( step != 0)
      {
         adjustTime(step);
         isTimeAdjusted = true; // Для извещения пользователя
                                // об изменении значения времени
         digitalClockDisplay(); // Обновляем отображаемое значение времени
         delay(100);
      }
   }
   return isTimeAdjusted;
}
```

Во всех предыдущих примерах значение времени отображается в окне монитора порта, но его также можно отображать и на ЖКД или светодиодном дисплее. Про-

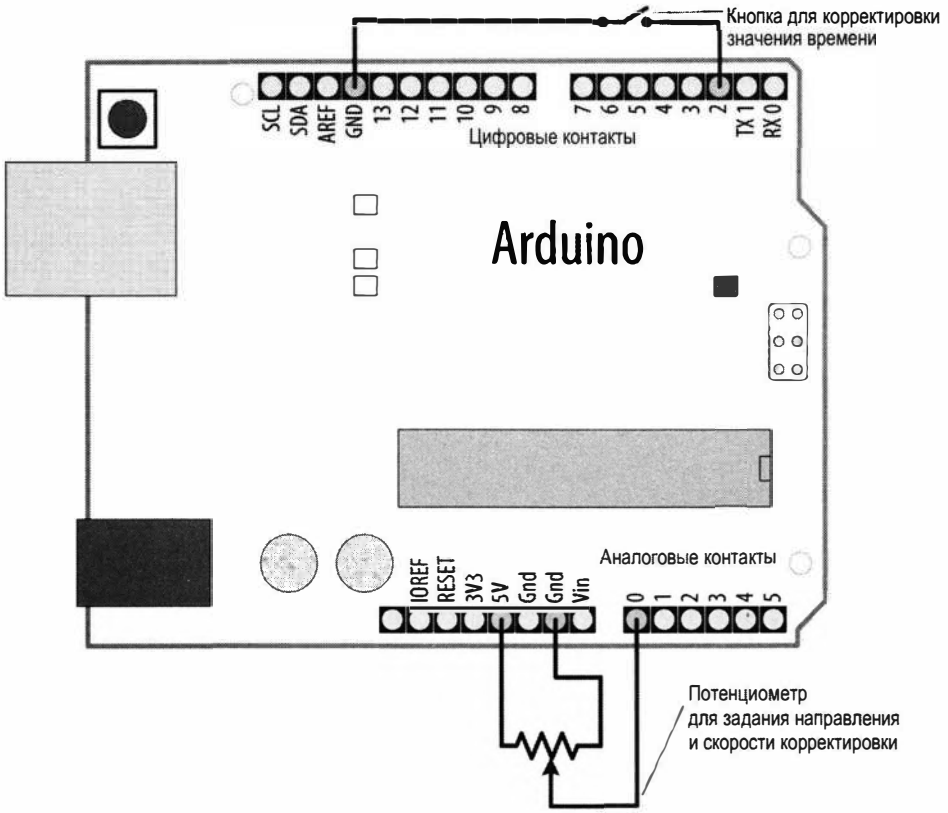


Рис. 12.2. Подключение потенциометра и кнопки для установки значения времени

граммное обеспечение для графических ЖК-дисплеев, рассматриваемое в *разд. 11.9* с предоставлением ссылок для его загрузки, включает несколько скетчей примеров для установки и отображения времени на стрелочных часах, прорисовываемых на ЖК-дисплее.

Библиотека TimeLib содержит несколько полезных функций для преобразования между разными временны́ми форматами и для других операций со временем. С помощью этих функций можно, например, узнать, сколько времени истекло с начала дня или сколько времени остается до конца дня.

Полный список этих функций можно найти в файле TimeLib.h, расположенном в папке libraries. Более подробная информация по работе с этими функциями приводится в *главе 16*, а в листинге 12.11 показано несколько таких функций с кратким описанием их назначения.

Листинг 12.11. Несколько функций для выполнения разных операций со временем

```
dayOfWeek( now() );          // Возвращает текущий день недели
                             // (Воскресенье — день 1)
elapsedSecsToday( now() );   // Возвращает количество секунд,
                             // истекших с начала текущего дня
```

```
nextMidnight( now() );          // Возвращает время, оставшееся до конца дня
elapsedSecsThisWeek( now() ); // Возвращает время, истекшее с начала текущей недели
```

Кроме отображения времени, можно также выводить текстовые строки названий дней и месяцев. В листинге 12.12 приводится вариант функции `digitalClockDisplay()`, которая выводит название дня и месяца.

Листинг 12.12. Вариант функции `digitalClockDisplay()`, выводящей название дня и месяца

```
void digitalClockDisplay()
{
   String timestr = String(hour()) + ":" + padDigits(minute()) +
                                     ":" + padDigits(second());
   Serial.println(timestr);
   String datestr = String(dayStr(weekday())) + ", " +
   String(monthShortStr(month())) + " " + String(year());
   Serial.println(datestr);
}
```

Дополнительная информация

Дополнительная информация по работе с библиотекой TimeLib приводится на ее веб-странице справки сайта Arduino (**https://oreil.ly/XxRdv**).

Дополнительная информация по времени UNIX содержится в статье на Wikipedia (**https://oreil.ly/w3xpW**).

Для получения времени UNIX и выполнения различных преобразований формата и операций со временем можно воспользоваться двумя популярными инструментами: Epoch Converter (**http://www.epochconverter.com**) и OnlineConversion.com (**https://oreil.ly/1PF2b**).

12.5. Создание события для периодического вызова функции

ЗАДАЧА

Требуется выполнять некоторое действие в определенное время по определенным дням.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 12.13. В скетче используется библиотека TimeAlarms — спутник библиотеки TimeLib, рассматриваемой в *разд. 12.4*. Эта библиотека позволяет с легкостью создавать часовые и дневные сигнальные события. Установите эту библиотеку с помощью Менеджера библиотек среды Arduino IDE (заодно установите и библиотеку TimeLib, если она еще не установлена).

Листинг 12.13. Создание сигнальных событий с помощью библиотеки TimeAlarms

```
/*
* Скетч TimeAlarmsExample
*
* Скетч вызывает сигнальные функции в 8:30 утра и в 17:45
* и эмулирует включение освещения вечером и выключение его утром
*
* Один таймер вызывается каждые 15 секунд
* Другой таймер вызывается только один раз после 10 секунд
*
* Начальное время устанавливается на 12:00 1 января 2020 г.
*/

#include <TimeLib.h>
#include <TimeAlarms.h>

void setup()
{
   Serial.begin(9600);
   while(!Serial);
   Serial.println("TimeAlarms Example");
   Serial.println("Alarms are triggered daily at 8:30 am and 17:45 pm");
          // Сигнальные события активируются каждый день в 8:30 и 17:45

   Serial.println("One timer is triggered every 15 seconds");
              // Один таймер активируется периодически каждые 15 секунд
   Serial.println("Another timer is set to trigger only once after 10 seconds");
             // Другой таймер вызывается только один раз после 10 секунд
   Serial.println();
   setTime(8,29,40,1,1,2020); // Устанавливаем начальное время
                              // на 8:29:40 am 1 января 2020 г.
   Alarm.alarmRepeat(8,30,0, MorningAlarm); // 8:30 каждый день
   Alarm.alarmRepeat(17,45,0,EveningAlarm); // 17:45 каждый день
   Alarm.timerRepeat(15, RepeatTask); // Таймер вызывается каждые 15 секунд
   Alarm.timerOnce(10, OnceOnlyTask); // Таймер вызывается один раз
                                      // после 10 секунд
}

void MorningAlarm()
{
   Serial.println("Alarm: - turn lights off");
                  // Сигнальное событие: — выключить освещение
}

void EveningAlarm()
{
```

```
   Serial.println("Alarm: - turn lights on");
                  // Сигнальное событие: – включить освещение
}

void RepeatTask()
{
   Serial.println("15 second timer"); // 15-секундный таймер
}

void OnceOnlyTask()
{
   Serial.println("This timer only triggers once");
                  // Этот таймер активируется только один раз
}

void loop()
{
   digitalClockDisplay();
   Alarm.delay(1000); // Пауза в 1 секунду между отображениями
                      // обновленного значения времени
}

// Дополняем числа ведущим 0
String padDigits(int digit)
{
   String str = String("0") + digit;       // Вставляем 0 перед цифрой
   return str.substring(str.length() - 2); // Удаляем все символы,
                                           // кроме двух последних
}

void digitalClockDisplay()
{
   String timestr = String(hour()) + ":" + padDigits(minute()) +
                                     ":" + padDigits(second());
   Serial.println(timestr);
   String datestr = String(year()) + "-" + padDigits(month()) +
                                     "-" + padDigits(day());
   Serial.println(datestr);
}
```

Обсуждение работы решения и возможных проблем

События можно планировать для активирования в определенное время дня (такие события называются *сигнальными*) или после истечения определенного периода времени (такие события называются *таймерами*). Каждое из этих типов событий можно запланировать для одноразового или периодического активирования.

Сигнальное событие можно запланировать для периодического активирования в определенное время суток следующим способом:

```
Alarm.alarmRepeat(8,30,0, MorningAlarm);
```

Этот код вызывает функцию `MorningAlarm` в 8:30 утра каждого дня.

Для активирования события только один раз можно использовать функцию `alarmOnce()`:

```
Alarm.alarmOnce(8,30,0, MorningAlarm);
```

Это сигнальное событие вызывает функцию `MorningAlarm` только один раз (в 8:30 утра) и больше не активируется.

В отличие от сигнальных событий, таймеры активируются не в определенное время суток, а после истечения заданного интервала времени. Интервал для таймеров можно задавать в виде любого количества секунд или часов, минут и секунд:

```
Alarm.timerRepeat(15, Repeats); // Таймер активируется каждые 15 секунд
```

Этот код вызывает функцию `Repeats` каждые 15 секунд.

Для одноразового активирования таймера используется функция `timeOnce()`:

```
Alarm.timerOnce(10, OnceOnly); // Таймер вызывается один раз после истечения 10 секунд
```

Этот код вызывает функцию `onceOnly()` по истечении 10 секунд после создания таймера.

Скетч должен регулярно вызывать функцию `Alarm.delay()`, т. к. эта функция проверяет состояние всех запланированных событий. Если этого не делать, то сигнальные события активироваться не будут. Когда требуется обслужить планировщик без задержек, функция вызывается с передачей ей параметра `0`: `Alarm.delay(0)`. При использовании в скетче библиотеки TimeAlarms всегда следует пользоваться функцией `Alarm.delay()`, а не просто `delay()`.

Для правильной работы библиотеки TimeAlarms необходимо также установить библиотеку TimeLib (см. *разд. 12.4*). Работа с библиотекой TimeAlarms не требует никакого дополнительного внешнего или встроенного оборудования. Планировщик не использует прерываний, поэтому для обработки сигнальных событий применяются такие же функции, что и любые другие функции скетча. Функции обработки событий прерываний имеют определенные ограничения, которые рассматриваются в *главе 18*, но они не распространяются на функции, используемые с библиотекой TimeAlarms.

Длительность интервалов таймеров может быть от одной секунды до нескольких лет. Для интервалов таймеров короче чем одна секунда лучше использовать библиотеку Tasker (**https://oreil.ly/mD6BF**).

Задачи планируются на исполнение в конкретные точки времени, обозначенные системными часами библиотеки TimeLib (более подробная информация об этом приводится в *разд. 12.4*). Если изменить системное время (например, вызвав функцию `setTime()`), времена активирования событий соответствующим образом не изменятся. Иными словами, если перевести системное время на один час вперед, то

все запланированные сигнальные события и таймеры будут происходить на один час раньше. Например, если в 1:00 установить активирование сигнального события на 3:00, а затем перевести системное время на 2:00, то запланированное событие активируется через один час. А если системные часы перевести на один час назад — на 12:00, то событие активируется через три часа, т. е. когда время по системным часам будет 3:00. Если перевести системное время на время более позднее, чем время активирования сигнального события, событие будет активировано немедленно (в действительности при следующем вызове функции `Alarm.delay()`).

Это ожидаемое поведение сигнальных событий: задачи планируются на исполнение в конкретное время суток и будут активированы в это время, — но эффект перевода системных часов на таймеры может быть не столь ясен. Если активирование таймера должно произойти через пять минут, но перед этим перевести системные часы назад на один час, таймер активируется только через один час и пять минут. Это относится и к периодическим таймерам, поскольку повторное активирование не будет запланировано, пока не будет выполнено предыдущее активирование.

Можно запланировать одновременно шесть сигнальных событий и таймеров. Но библиотеку можно модифицировать для планирования большего количества заданий (о том, как это сделать, подробно рассказано в *разд. 16.3*).

Одноразовые события и таймеры высвобождаются после активирования, и их можно планировать снова при условии, что общее количество событий и таймеров, ожидающих активирования, не превышает шести. В листинге 12.14 приводится фрагмент кода, демонстрирующий, как можно повторно планировать одноразовый таймер.

Листинг 12.14. Повторное планирование одноразового таймера

```
Alarm.timerOnce(random(10), randomTimer); // Активируется после
                            // произвольного количества секунд
void randomTimer()
{
   int period = random(2,10);      // Получаем новый период произвольной длительности
   Alarm.timerOnce(period, randomTimer); // Активируется после второго
                                         // периода произвольной длительности
}
```

12.6. Работа с часами реального времени

ЗАДАЧА

Требуется разработать приложение, в котором используется время, предоставляемое часами RTC (Real Time Clock, часы реального времени), например, на основе микросхемы DS1307. Модули часов RTC обычно оснащены батарейкой для резервного питания, поэтому предоставляемое ими время будет правильным даже после сброса платы Arduino или выключения питания.

РЕШЕНИЕ

Самый простой способ использования часов RTC — задействовать библиотеку DS1307RTC.h (библиотеку — спутник библиотеки TimeLib). Установите эту библиотеку с помощью Менеджера библиотек среды Arduino IDE (заодно установите и библиотеку TimeLib, если она еще не установлена). В листинге 12.15 приводится скетч для работы с часами RTC, использующий эту библиотеку. Скетч будет работать с часами RTC как на основе микросхемы DS1307, так и на основе микросхемы DS1337.

Листинг 12.15. Скетч для работы с часами RTC, использующий библиотеку DS1307RTC.h

```
/*
* Скетч TimeRTC
* Демонстрация работы с часами RTC, используя библиотеку TimeLib
*
*/

#include <TimeLib.h>
#include <Wire.h>
#include <DS1307RTC.h> // Базовая библиотека DS1307, возвращающая время
                       // в виде time_t

void setup()
{
   Serial.begin(9600);
   while(!Serial);       // Для плат Leonardo и других 32-разрядных плат
   setSyncProvider(RTC.get); // Функция, получающая время от часов RTC
   if(timeStatus()!= timeSet)
      Serial.println("Unable to sync with the RTC");
                       // Не удается синхронизироваться с часами RTC
   else
      Serial.println("RTC has set the system time");
                       // Системное время было установлено по часам RTC
}

void loop()
{
   digitalClockDisplay();
   delay(1000);
}

// Pad digits with a leading 0
String padDigits(int digit)
{
   String str = String("0") + digit;       // Вставляем 0 перед цифрой
```

```
    return str.substring(str.length() - 2); // Удаляем все символы,
                                            // кроме двух последних
}

void digitalClockDisplay()
{
    String timestr = String(hour()) + ":" + padDigits(minute()) +
                                      ":" + padDigits(second());
    Serial.println(timestr);
String datestr = String(year()) + "-" + padDigits(month()) +
                                    "-" + padDigits(day());
Serial.println(datestr);
}
```

Большинство модулей часов RTC для Arduino используют для взаимодействия с платой протокол I^2C (подробно об этом протоколе рассказано в *главе 13*). Подключение модуля часов RTC к плате Arduino показано на рис. 12.3. Контакт SCL (или Clock) модуля подключается к контакту A5 платы Arduino, а контакт SDA (или Data) — к контакту A4 (для работы по протоколу I^2C используются аналоговые контакты A4 и A5 платы Arduino — см. *главу 13*). Будьте внимательны при подключении питания модуля часов RTC, чтобы случайно не перепутать контакты питания.

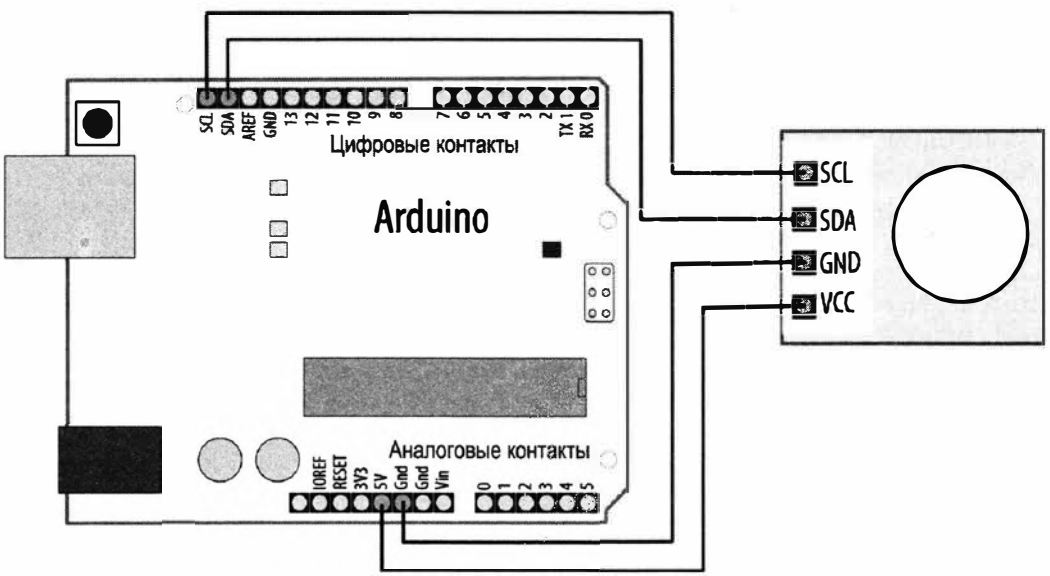


Рис. 12.3. Подключение модуля часов RTC к плате Arduino

Обсуждение работы решения и возможных проблем

Код скетча решения похож на код предыдущих скетчей, использующих библиотеку TimeLib, с тем исключением, что значение времени предоставляется часами RTC,

а не передается по последовательному интерфейсу или жестко задается в коде. Для этого требуется единственная дополнительная строка кода:

```
setSyncProvider(RTC.get); // Функция, получающая время от часов RTC
```

Функция `setSyncProvider()` дает указание библиотеке TimeLib, каким образом получать информацию для установки и обновления времени. Метод (функция) `RTC.get()` библиотеки RTC возвращает текущее время в формате, используемом библиотекой TimeLib (время UNIX).

При каждом начале работы платы Arduino функция `setup()` будет вызывать функцию `RTC.get()`, чтобы получить время с модуля часов RTC.

Но прежде чем часы RTC можно будет использовать в качестве источника текущего времени, на них необходимо настроить правильное время. В листинге 12.16 приводится скетч, который позволяет устанавливать время на часах RTC. Исполнять его нужно только один раз после подключения батарейки к модулю часов RTC, замены батарейки или при смене времени.

Листинг 12.16. Скетч для установки времени на часах RTC

```
/*
* Скетч Set RTC time
* Устанавливает часы RTC через последовательный интерфейс.
  Упрощенная версия скетча примера TimeSerial
* из библиотеки TimeLib.
*
* Устанавливаем время, отправляя скетчу монитора порта строку,
  начинающуюся буквой T, за которой следует 10 цифр, обозначающих
  количество секунд, истекших с 1 января 1970 г.
  Например, строка T1569888000 представляет 12:00 дня 1 октября 2019 г.
*/

#include <TimeLib.h>
#include <Wire.h>
#include <DS1307RTC.h> // Базовая библиотека DS1307, возвращающая время
                       // в виде time_t

void setup()
{
   Serial.begin(9600);
   setSyncProvider(RTC.get); // Функция, получающая время от часов RTC
   if(timeStatus()!= timeSet)
      Serial.println("Unable to sync with the RTC");
                       // Не удается синхронизироваться с часами RTC
   else
      Serial.println("RTC has set the system time");
                       // Системное время было установлено по часам RTC
}
```

```
void loop()
{
   if(Serial.available())
   {
      processSyncMessage();
   }
   digitalClockDisplay();
   delay(1000);
}

// Дополняем числа ведущим 0
String padDigits(int digit)
{
   String str = String("0") + digit; // Вставляем 0 перед цифрой
   return str.substring(str.length() - 2); // Удаляем все символы,
                                           // кроме двух последних
}

void digitalClockDisplay()
{
   String timestr = String(hour()) + ":" + padDigits(minute()) +
                                     ":" + padDigits(second());
   Serial.println(timestr);
   String datestr = String(year()) + "-" + padDigits(month()) +
                                     "-" + padDigits(day());
   Serial.println(datestr);
}

#define TIME_HEADER 'T' // Ярлык заголовка для сообщения синхронизации
             // времени, передаваемого по последовательному интерфейсу

// Выполняем парсинг сообщения синхронизации
void processSyncMessage()
{
   time_t pctime = 0;

   if(Serial.find(TIME_HEADER))
   {
      pctime = Serial.parseInt();
      setTime(pctime); // Устанавливаем часы системы на время,
                       // полученное через последовательный интерфейс
      RTC.set(pctime); // Также устанавливаем время на часах RTC
   }
}
```

Это почти такой же скетч, что и скетч TimeSerial из *разд. 12.4* для установки времени системы по значению, получаемому через последовательный интерфейс

(см. листинг 12.8), с той лишь разницей, что здесь при установке времени системы по полученному с компьютера сообщению со значением времени также вызывается функция `RTC.set()` для установки часов RTC:

```
setTime(pctime); // Устанавливаем часы системы на время, полученное
                 // через последовательный интерфейс
RTC.set(pctime); // Также устанавливаем время на часах RTC
```

Модуль часов RTC взаимодействует с платой Arduino по интерфейсу I^2C, который подробно рассматривается в *главе 13*.

Время часов RTC можно задать по времени компиляции скетча оператором:

```
rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
```

предварительно установив библиотеку Adafruit_RTCLib. Более подробная информация на эту тему приводится на странице: **https://oreil.ly/j0yYr** веб-сайта компании Adafruit.

Некоторые из более новых плат Arduino оснащены встроенными часами RTC, поддержка которых обеспечивается библиотекой RTC (**https://oreil.ly/sifTk**). Такой возможностью обладают например, платы Zero (**https://oreil.ly/zo71v**), MKRZero (**https://oreil.ly/t_ckx**) и MKR1000 (**https://oreil.ly/Aawqq**). Но для правильной работы этих часов необходимо использовать батарейку резервного питания для их работы при отключенном питании платы.

Дополнительная информация

Модуль часов RTC предлагается компанией SparkFun (артикул BOB-00099) (**https://oreil.ly/9vYtm**).

Компания Adafruit также предлагает модуль часов RTC нам микросхеме DS1307 (артикул 3296) (**https://oreil.ly/CAqr-**).

ГЛАВА 13

Протоколы связи I²C и SPI

13.0. Введение

Целью разработки протоколов связи I²C (Inter-Integrated Circuit — межсхемный интерфейс интегральных схем) и SPI (Serial Peripheral Interface — последовательный синхронный периферийный интерфейс) было создание простого способа организации обмена информацией между датчиками и микроконтроллерами, такими как Arduino. Использование обоих этих протоколов на платформе Arduino не представляет никаких сложностей благодаря наличию соответствующих библиотек.

Выбор между этими протоколами обычно определяется устройством, которое требуется подключить к плате Arduino, — например, датчиком, приводом, другой платой и т. п. Некоторые устройства поддерживают оба протокола, но обычно устройство или микросхема может работать только с одним из них.

Преимущество протокола I²C состоит в том, что для него требуется только две сигнальные линии (данных и тактирования), тогда как для протокола SPI их надо уже четыре. Еще одним преимуществом работы по протоколу I²C является подтверждение правильности полученных сигналов. К недостаткам этого протокола относится более низкая скорость передача данных, чем у протокола SPI, а также симплексный режим связи, когда данные одновременно могут передаваться только в одном направлении, в результате чего скорость связи падает вдвое по сравнению с дуплексным режимом связи, при котором данные могут одновременно передаваться в обоих направлениях. Кроме того, для обеспечения надежной связи к сигнальным линиям протокола I²C необходимо подключать повышающие резисторы (их использование подробно рассматривается в *главе 5*). Точное значение номинала таких повышающих резисторов зависит от нескольких факторов, одним из которых является длина проводов подключения. Но по большому счету лучше всего, наверное, подойдет номинал 4,7 кОм.

При использовании устройства I²C, смонтированного на адаптерной плате, подключать повышающие резисторы может и не потребоваться, поскольку производители иногда устанавливают их на адаптерной плате. Но чтобы знать это наверняка, необходимо уточнить их наличие по справочному листку на устройство. Например, на рис. 13.1 показан фрагмент принципиальной схемы адаптерной платы контроллера светодиодной матрицы 16×8 HT16K33 с интерфейсом I²C компании Adafruit (артикул 1427), где можно четко видеть повышающие резисторы.

Компания Adafruit устанавливает повышающие резисторы номиналом 10 кОм на всех своих платах, поскольку убедилась, что это значение хорошо работает

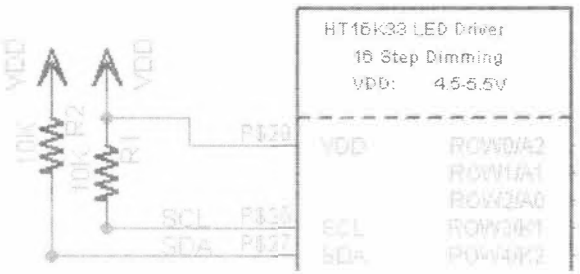


Рис. 13.1. Повышающие резисторы устройства с интерфейсом I^2C, установленного на адаптерной плате

в реальных условиях. Теперь, если подключить две такие адаптерные платы, у нас получится два резистора номиналом 10 кОм, включенные параллельно. Общее сопротивление этих резисторов вычисляется по формуле параллельных сопротивлений:

$$(10 \text{ кОм} \times 10 \text{ кОм}) / (10 \text{ кОм} + 10 \text{ кОм}) = 5 \text{ кОм},$$

что уже близко к ранее упомянутому номиналу 4,7 кОм. При подключении трех устройств общее сопротивление трех параллельных повышающих резисторов станет уже:

$$1/(1/10 \text{ кОм} + 1/10 \text{ кОм} + 1/10 \text{ кОм}) = 3{,}3 \text{ кОм},$$

что все еще остается в общепринятых пределах значения сопротивления повышающих резисторов для сигнальных линий интерфейса I^2C. Интересное обсуждение значений сопротивлений повышающих резисторов для сигнальных линий интерфейса I^2C приводится в статье Ника Гаммона (Nick Gammon) (**https://oreil.ly/iZKPG**).

Достоинством интерфейса SPI является более высокая скорость передачи данных, а также наличие отдельных линий ввода и вывода, что позволяет одновременно вести и прием, и передачу. Кроме двух линий данных для каждого устройства на шине SPI требуется линия для сигнала тактирования, а также отдельная линия для выбора этого устройства. Таким образом, минимальное количество сигнальных линий при работе по этому интерфейсу равно четырем. В результате при подключении нескольких устройств количество соединений может превысить предел, с которым удобно работать.

В большинстве проектов Arduino устройства SPI используются для приложений с высокой скоростью передачи данных — например, работающих с платами памяти и адаптерами Ethernet. При этом подключается только одно такое устройство. Протокол I^2C обычно чаще задействуется для подключения датчиков, которые не отправляют большого объема данных.

В этой главе показано использование шин I^2C и SPI для работы с периферийными устройствами, а также применение шины I^2C для подключения двух или более плат Arduino в многоплатных проектах. Но прежде чем приступать к рассмотрению решений, познакомимся с некоторой предварительной информацией о протоколах I^2C

и SPI, а также разберемся с вопросами совместной работы устройств с рабочим напряжением 3,3 В и плат с рабочим напряжением 5 В.

Интерфейс I²C

Две сигнальные линии интерфейса I²C носят названия SCL (сигнал тактирования) и SDA (передача данных). Эти линии в платах Arduino Uno и Zero, а также в совместимых платах, выводятся на соответствующие контакты SCL и SDA (см. *разд. 1.2*). Плата Arduino Nano, а также более ранние платы Uno не имеют отдельных контактов для линий SCL и SDA, поэтому при работе с ними мы воспользуемся аналоговыми контактами: A5 — для линии SCL и A4 — для линии SDA. На плате Mega контакт 20 отводится для линии SDA, а контакт 21 — для линии SCL. При использовании плат с другими форм-факторами — например, платы Teensy компании PJRC или платы Feather компании Adafruit — за информацией о номерах контактов для этих сигнальных линий обратитесь к справочному листку и/или другой документации на имеющуюся плату.

Одно из подключенных к шине I²C устройств является *ведущим* (или основным). Его задачей является координация обмена информацией между другими подключенными устройствами, которые называются *ведомыми*. Ведущим на шине I²C может быть только одно устройство. В большинстве случаев таким устройством является плата Arduino, которая управляет другими подключенными к шине устройствами. На рис. 13.2 показана организация шины I²C с одним ведущим устройством и несколькими ведомыми.

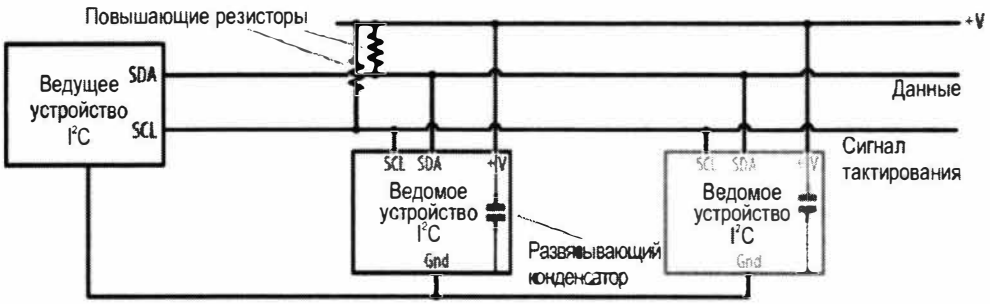


Рис. 13.2. Одно ведущее устройство шины I²C координирует работу всех ведомых устройств

Для правильной работы устройств I²C их шины «земли» должны быть соединены вместе. Это можно сделать, соединив шины «земли» каждого подключенного к плате Arduino устройства I²C с ее контактом GND.

Для идентификации каждому ведомому устройству присваивается уникальный адрес. Некоторые устройства I²C имеют фиксированный адрес (например, контроллер Wii Nunchuk, рассматриваемый в *разд. 13.6*), тогда как адреса других устройств можно настраивать установкой комбинаций уровней на их адресных контактах (см. *разд. 13.4*) или отправкой команд инициализации.

Для указания адресов I²C в скетчах Arduino используются 7-разрядные двоичные значения. В некоторых справочных листках приводятся 8-разрядные значения. Тогда, чтобы получить правильное 7-разрядное значение, 8-разрядное значение нужно разделить на 2.

Протоколы I²C и SPI определяют только способ обмена сообщениями между устройствами. Сами же сообщения зависят от каждого конкретного устройства и его назначения. Определить команды, необходимые для управления конкретным устройством I²C, и требуемые или возвращаемые им данные можно по справочному листку на него.

Библиотека Wire Arduino скрывает от пользователя всю низкоуровневую функциональность протокола I²C и позволять использовать простые команды для инициализации устройств и обмена данными с ними.

Корректировка устаревшего кода библиотеки Wire для работы с Arduino IDE 1.0 и более поздних версий

В версии Arduino IDE 1.0 библиотека Wire подверглась некоторым изменениям, и чтобы скетчи предыдущих версий среды Arduino IDE компилировались в версии 1.0, их необходимо должным образом откорректировать. В частности, методы (функции) `send()` и `receive()` были переименованы, чтобы обеспечить единообразие терминологии с другими библиотеками:

- замените `Wire.send()` на `Wire.write()`
- замените `Wire.receive()` на `Wire.read()`.

Необходимо также указывать тип переменной для записываемых констант аргументов. Например, замените:

`Wire.write(0x10)` на `Wire.write((byte)0x10)`

Использование устройств с напряжением питания 3,3 В совместно с платами с напряжением питания 5 В

Многие устройства I²C предназначены для работы с логическими уровнями с напряжением величиной 3,3 В, и подключение таких устройств к плате Arduino с напряжением питания и соответственно логическими уровнями величиной 5 В может вывести их из строя. Для безопасного подключения таких устройств можно использовать модуль двунаправленного преобразования логических уровней — например, модуль BOB-12009 компании Adafruit (артикул 757). Схема подключения устройств к такому модулю показана на рис. 13.3. С одной стороны платы модуля расположены контакты для подключения сигналов более низкого напряжения (LV, low voltage) 3,3 В, а с другой — сигналов более высокого напряжения (HV, high voltage) 5 В.

Устройство I²C с логическими уровнями напряжением 3,3 В подключается к контактам стороны LV следующим образом:

- контакт LV1 (контакт A1 на плате модуля BOB-12009) подключается к контакту SDA низковольтного устройства I²C;

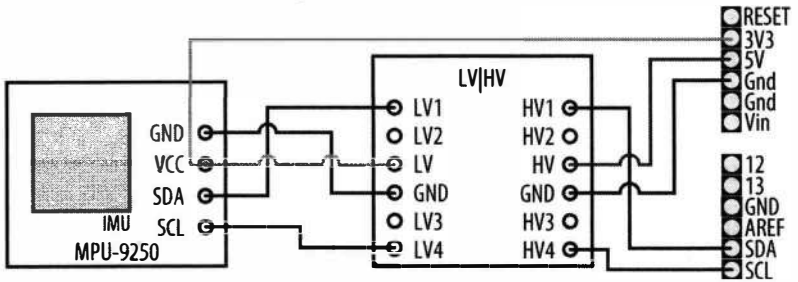


Рис. 13.3. Подключение низковольтного устройства к плате Arduino с помощью модуля преобразования логических уровней

- контакт LV2 (контакт A2 на плате модуля BOB-12009) подключается к контакту SCL низковольтного устройства I²C;
- контакт LV преобразователя подключается к контакту питания низковольтного устройства I²C, а также к источнику питания напряжением 3,3 В — например, к контакту 3V3 платы Arduino;
- контакт GND преобразователя подключается к контакту GND низковольтного устройства.

Высоковольтное устройство — например, плата Arduino Uno — подключается к преобразователю логических уровней следующим образом:

- контакт HV1 (контакт B1 на плате модуля BOB-12009) подключается к контакту SDA высоковольтного устройства I²C;
- контакт HV2 (контакт B2 на плате модуля BOB-12009) подключается к контакту SCL низковольтного устройства I²C;
- контакт HV преобразователя подключается к контакту питания высоковольтного устройства, т. е. к контакту 5V в случае платы Arduino Uno;
- контакт GND преобразователя подключается к контакту GND высоковольтного устройства.

К одному модулю преобразователя уровней можно подключать несколько устройств I²C, как показано на рис. 13.4.

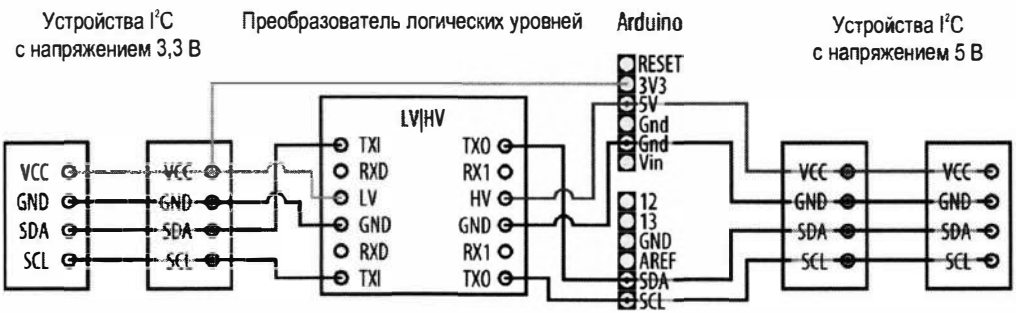


Рис. 13.4. Подключение нескольких низковольтных и высоковольтных устройств I2C к одному преобразователю уровней

Примеры с использованием модуля преобразователя логических уровней с платами Arduino с питанием 5 В приводятся в *разд. 6.15–6.17*, где рассматривается работа с модулем акселерометра MPU-9250 с напряжением питания 3,3 В.

Интерфейс SPI

В состав среды Arduino IDE входит библиотека, позволяющая организовать взаимодействие с устройствами SPI. Устройства SPI имеют отдельные линии для приема и передачи данных (обозначенные MOSI и MISO), а также линию сигнала тактирования. Все эти три сигнальные линии ведущего устройства подключаются к соответствующим линиям одного или нескольких ведомых устройств. Для выбора конкретного ведомого устройства служит линия SS (Slave Select — выбор ведомого), которая также может называться CS (Chip Select — выбор микросхемы). Подключение ведущего и ведомых устройств SPI показано на рис. 13.5.

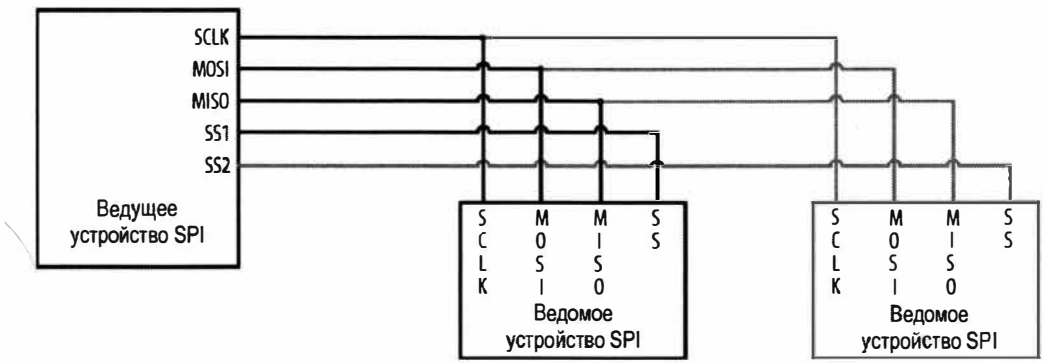


Рис. 13.5. Подключение ведущего и ведомых устройств SPI

В табл. 13.1 показаны номера контактов плат Arduino Uno и Mega для использования с аппаратным интерфейсом SPI.

Таблица 13.1. *Цифровые контакты плат Arduino, используемые для работы с аппаратным интерфейсом SPI*

Сигнал SPI	Плата Arduino Uno	Плата Arduino Mega
SCLK (Сигнал тактирования)	13	52
MISO (Исходящие данные)	12	50
MOSI (Входящие данные)	11	51
SS/CS (Выбор ведомого (микросхемы))	10	53

В некоторых платах с микроконтроллером SAMD (например, в платах Arduino Zero, платах MO Express компании Adafruit и платах RedBoard Turbo компании SparkFun) сигналы интерфейса SPI выводятся только на контакты разъема ICSP. В некоторых 8-разрядных платах (например, Leonardo) сигналы SPI также выводят-

ся только на разъем ICSP. Вывод конкретных сигналов SPI на контакты этого разъема показан на рис. 13.6. При этом в таких платах для сигнала SS (выбор ведомого) можно использовать любой цифровой контакт. Часто для этого используют контакт 10. Но в случае подключения к шине SPI нескольких устройств (где все они используют общие линии SCL, MISO и MOSI), для каждого из них на плате Arduino нужно выделить отдельный контакт выбора устройства.

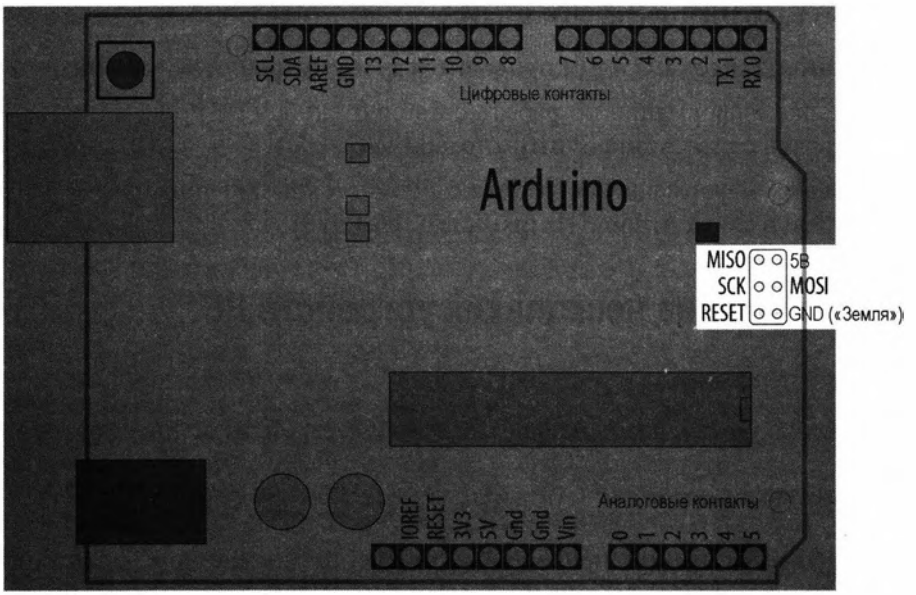


Рис. 13.6. Вывод сигналов SPI на разъем ICSP платы Arduino

Вам могут повстречаться некоторые библиотеки, позволяющие использовать *программный* интерфейс SPI. Принцип работы программного SPI похож на принцип работы программного последовательного интерфейса Serial (см. *разд. «Эмуляция аппаратного последовательного порта посредством цифровых контактов ввода/вывода» главы 4*) в том смысле, что аппаратное обеспечение интерфейса SPI не используется, а все операции по SPI-обмену осуществляются программным способом. Подобно программному последовательному интерфейсу, скорость обмена по программному интерфейсу SPI будет более медленной, чем по аппаратному, а также могут присутствовать некоторые другие ограничения. Но если по какой-либо причине использовать аппаратный интерфейс SPI не представляется возможным, альтернатива в виде программного варианта этого интерфейса может быть очень полезной. Наличие программного интерфейса SPI можно определить по наличию двух форм конструктора. Конструктору аппаратного интерфейса SPI передается только номер контакта для сигнала SS выбора устройства, поскольку номера контактов для остальных трех сигналов определяются используемой платой Arduino. А конструктору программного интерфейса SPI передаются номера контактов для всех четырех сигналов.

На некоторых устройствах SPI (например, на цветном ЖКД ST7*xx*, рассматриваемом в *разд. 11.10*) присутствует дополнительный контакт TFT_DC для сигнала переключения между режимом передачи данных и режимом передачи команд. Для ЖК-дисплея ST77*xx* в конструкторе также можно указать номер контакта сброса, но эта возможность не является частью протокола SPI. Таким образом, конструктор

аппаратного интерфейса SPI для ЖК-дисплея ST77*xx* будет выглядеть следующим образом:

```
Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);
```

А конструктор программного интерфейса SPI для ЖК-дисплея ST77*xx* так:

```
Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK, TFT_RST);
```

Дополнительная информация

Сравнение интерфейсов I^2C и SPI приводится здесь: **https://oreil.ly/PJ0kz**.

Дополнительная информация по работе с библиотекой Wire приводится на ее веб-странице справки сайта Arduino *(***https://oreil.ly/eQ9UY**).

Дополнительная информация по работе с библиотекой SPI приводится на ее веб-странице справки сайта Arduino (**https://oreil.ly/aRJrd**).

13.1. Подключение нескольких устройств I^2C

ЗАДАЧА

Требуется подключить к плате Arduino несколько устройств по интерфейсу I^2C.

РЕШЕНИЕ

Эта задача решается схемой, приведенной на рис. 13.7, и скетчем из листинга 13.1, который работает с этой схемой. В схеме используется датчик качества воздуха, с помощью которого скетч измеряет общую концентрацию летучих органических

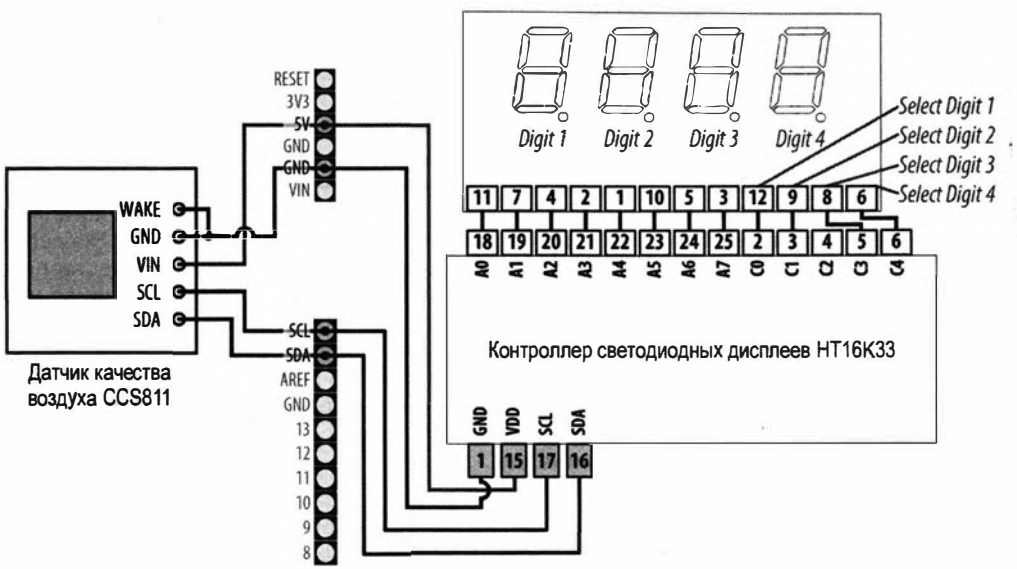


Рис. 13.7. Подключение датчика качества воздуха и светодиодного дисплея к плате Arduino

соединений в миллиардных долях и отображает результат на четырехразрядном светодиодном дисплее. Как датчик качества воздуха, так и контроллер светодиодного дисплея взаимодействуют с платой Arduino посредством интерфейса I²C. На рис. 13.7 показана схема подключения этих двух устройств к плате Arduino, а также подключение к ней четырехразрядного светодиодного дисплея.

Листинг 13.1. Скетч для взаимодействия с двумя устройствами по интерфейсу I²C

```
/*
* Скетч Two I2C Device
* Считывает выходной сигнал датчика качества воздуха и отображает
  полученное значение на светодиодном дисплее
*/

#include <Adafruit_CCS811.h>
#include <Adafruit_GFX.h>
#include <Adafruit_LEDBackpack.h>

// Создаем экземпляры объектов для датчика и дисплея
Adafruit_CCS811 ccs;
Adafruit_7segment matrix = Adafruit_7segment();

void setup()
{
   Serial.begin(9600);
   if(!ccs.begin())
   {
      Serial.println("Could not start sensor.");
                    // Не удалось запустить датчик
      while(1); // Останов
   }
   while(!ccs.available()); // Дожидаемся готовности датчика
   matrix.begin(0x70); // Запускаем светодиодный дисплей
}

void loop()
{
   if(ccs.available())
   {
      if(!ccs.readData())
      {
         int tvoc = ccs.getTVOC(); // Получаем значение концентрации
                                   // летучих органических соединений
         matrix.println(tvoc);     // Записываем полученное значение на дисплей
         matrix.writeDisplay();    // Обновляем состояние дисплея
      }
   }
   delay(500);
}
```

Обсуждение работы решения и возможных проблем

В решении используются два устройства I^2C: датчик качества воздуха CCS811 компании ams и контроллер светодиодных дисплеев HT16K33 компании Holtek. Оба эти устройства предлагаются разными поставщиками уже смонтированными на адаптерных платах. В частности, компания Adafruit предлагает датчик качества воздуха как артикул 3566, а контроллер светодиодных дисплеев как артикул 1427. А компания SparkFun предлагает датчик качества воздуха как артикул SEN-14193. В решении используются устройства компании Adafruit, а также библиотеки Adafruit для работы с этими устройствами: библиотека Adafruit_CCS811 и библиотека Adafruit_LED_Backpack. Обе библиотеки устанавливается с помощью Менеджера библиотек среды Arduino IDE (подробно об установке библиотек рассказано в *главе 16*).

При подключении более чем одного устройства I^2C все линии SDA и SCL этих устройств соединяются вместе. На линиях питания каждого устройства необходимо установить блокирующие конденсаторы емкостью 0,1 мкФ, если только такие конденсаторы уже не установлены на адаптерных платах (это можно проверить в справочном листке или принципиальной схеме для используемой адаптерной платы). Отметим, что в нашем случае эти конденсаторы на адаптерных платах установлены. Линии «земли» устройств и платы Arduino необходимо соединить вместе, даже если устройства питаются от отдельных источников питания (например, батареек).

Если адаптерные платы устройств содержат повышающие резисторы для сигнальных линий интерфейса I^2C (SCL и SDA), в схему, возможно, не нужно будет добавлять внешние повышающие резисторы (см. *разд. 13.0*). Отметми, что адаптерные платы устройств компании Adafruit содержат такие повышающие резисторы, но в случае использования плат других поставщиков следует проверить наличие в них повышающих резисторов в справочном листке на используемые устройства.

Скетч инициализирует оба устройства в функции `setup()`, а в главном цикле `loop()` периодически считывает выходной сигнал датчика концентрации в воздухе летучих органических соединений, отображая на светодиодном дисплее каждое полученное значение. При использовании иного светодиодного дисплея, чем указанный в этом решении, он может иметь другую схему расположения контактов, поэтому сверьтесь со справочным листком на него и откорректируйте его подключение должным образом.

Оба устройства могут работать с напряжениями логических сигналов величиной 5 В, поэтому их можно напрямую подключать к платам Arduino с напряжением питания 5 В. При соединении датчика CCS811 с платой Arduino с напряжением питания 3,3 В питание на датчик следует подавать с контакта 3V3 платы, а не с контакта 5 В, поскольку в последнем случае напряжение логических сигналов I^2C будет слишком высоким для платы Arduino. А вот использовать контроллер светодиодного дисплея HT16K33 с платой Arduino с напряжением питания 3,3 В будет несколько сложнее, поскольку согласно спецификации на этот контроллер для его правильной работы требуется напряжение питания величиной минимум 4,5 В. Эту проблему можно было бы решить, задействовав преобразователь уровней логиче-

ских сигналов (см. *разд. «Использование устройств с напряжением питания 3,3 В с устройствами с напряжением питания 5 В»* ранее в этой главе), подключив плату Arduino или совместимую плату к его низковольтной стороне (LV). Но согласно опыту некоторых пользователей этот контроллер работает должным образом даже с напряжением питания величиной 3,3 В, так что можно сначала самому попробовать проверить эту возможность.

Контроллер светодиодных дисплеев HT16K33 рассчитан на работу с дисплеями с общим катодом. На рис. 13.7 показана распространенная схема расположения контактов для 4-разрядных 7-сегментных дисплеев. Контакты с A0 по A15 контроллера обычно используются для управления семью сегментами и десятичной точкой дисплея. В скетче решения задействованы только контакты с A0 по A7, однако при использовании матричного светодиодного дисплея контактов было бы занято больше. Контакты с C0 по C7 служат для выбора разряда. Контроллер поочередно активирует отображение каждого разряда, полагаясь на инерционность зрительного восприятия для создания впечатления одновременного включения всех разрядов.

Некоторые светодиодные четырехразрядные семисегментные дисплеи могут отображать между разрядами двоеточие, что позволяет использовать их для создания цифровых часов. Библиотека Adafruit_LED_Backpack полагает, что это двоеточие подключено к контакту C4 (физический контакт 4). Но в скетче решения двоеточие не используется, поэтому контакт C4 никуда не подключается.

Для взаимодействия с датчиком и контроллером дисплея библиотеки Adafruit полагаются на библиотеку Wire. Например, задать минимальную яркость дисплея можно, задав в методе (функции) `setBrightness()` объекта `matrix` библиотеки Adafruit_LED_Backpack значение 1 (максимальная яркость равна значению 15):

```
matrix.setBrightness(1);
```

Чтобы выполнить эту задачу, библиотека Adafruit_LED_Backpack выдаст следующий набор команд библиотеке Wire:

```
#define HT16K33_CMD_BRIGHTNESS 0xE0

Wire.beginTransmission(0x70);
Wire.write(HT16K33_CMD_BRIGHTNESS | 1);
Wire.endTransmission();
```

А в результате последовательного вызова методов `ccs.readData()` и `ccs.getTVOCQ()` создается набор команд, показанный в листинге 13.2 (где значение `0x5A` является адресом I²C датчика качества воздуха CCS811). Показания датчика формируются объединением третьего и четвертого байта (не забываем, что в языке C счет элементов массивов начинается с нуля) в слово.

Листинг 13.2. Получение показаний датчика качества воздуха

```
uint8_t buf[8];
Wire.beginTransmission(0x5A);
```

```
Wire.write(0x02);          // Записываем в регистр 0x02
Wire.endTransmission();

Wire.requestFrom(0x5A, 8); // Запрашиваем 8 байтов у датчика CCS811
for(int i=0; i < 8; i++)
{
   buf[i] = Wire.read();
}
int tvoc = word(buf[2], buf[3]);
```

Дополнительная информация

Дополнительная информация по работе с 7-сегментными светодиодными дисплеями приводится в *разд. 7.11*.

Подробная информация по контроллеру жидкокристаллического дисплея HT16K33 приводится в его справочном листке (**https://oreil.ly/yOFJ8**).

Подробная информация по датчику качества воздуха CCS811 приводится в его справочном листке (**https://oreil.ly/VMU-G**).

13.2. Подключение нескольких устройств SPI

ЗАДАЧА

Требуется подключить к плате Arduino несколько устройств по интерфейсу SPI.

РЕШЕНИЕ

Скетч в листинге 13.3 считывает растровые изображения с SD-карты и отображает их на TFT-дисплее. Как считыватель SD-карт, так и TFT-дисплей являются устройствами SPI. Подключение этих устройств к плате Arduino показано на рис. 13.8.

Листинг 13.3. Считывание изображения с SD-карты и вывод его на TFT-дисплей

```
/*
* Скетч Two SPI Device
* Считывает все растровые изображения, хранящиеся на SD-карте,
  и отображает их на TFT-дисплее
*/

#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <SdFat.h>
#include <Adafruit_ImageReader.h>

#define SD_CS 4    // Номер контакта для линии выбора устройства считывателя SD-карт
#define TFT_CS 10 // Номер контакта для линии выбора устройства TFT-дисплея
```

```
#define TFT_DC 9  // Номер контакта для линии данных
                  // и команд TFT-дисплея
#define TFT_RST 8 // Номер контакта для линии сброса TFT-дисплея

// Создаем экземпляры объектов для обоих устройств SPU
SdFat SD;
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);

SdFile root;                       // Считываем каталог SD-карты
Adafruit_ImageReader reader(SD); // Объект для загрузки и отображения изображения

void setup(void)
{
  Serial.begin(9600);
  if(!SD.begin(SD_CS, SD_SCK_MHZ(25))) // Инициализируем считыватель
                                       // SD-карт на частоте 25 МГц
  {
    Serial.println("Could not initialize SD card");
                  // Не удалось инициализировать считыватель SD-карт
    while(1); // Останов
  }

  tft.begin(); // Инициализируем TFT-дисплей

  if (!root.open("/"))
  {
    Serial.println("Could not read SD card directory");
                  // Не удалось прочитать каталог SD-карты
    while(1); // Останов
  }
}

void loop()
{
  ImageReturnCode rc; // Код возврата с операций по чтению изображений
  SdFile file;        // Текущий файл
  char filename[256]; // Буфер для названия файла

  while (file.openNext(&root, O_RDONLY)) // Находим следующий файл
                                         // на SD-карте
  {
    file.getName(filename, sizeof(filename)/ sizeof(filename[0]));
    if(isBMP(filename)) // Если это файл BMP, отображаем его
                        // на TFT-дисплее
    {
      tft.fillScreen(0);
      rc = reader.drawBMP(filename, tft, 0, 0);
```

```
            delay(2000); // Берем короткую паузу
        }
        file.close();
    }
    root.rewind(); // Возвращаемся к первому файлу в корневом каталоге
}

// Функция для определения, является ли файл растровым
// изображением (BMP)
int isBMP(char fname[])
{
    String fn = String(fname);
    fn.toLowerCase();
    return fn.endsWith("bmp");
}
```

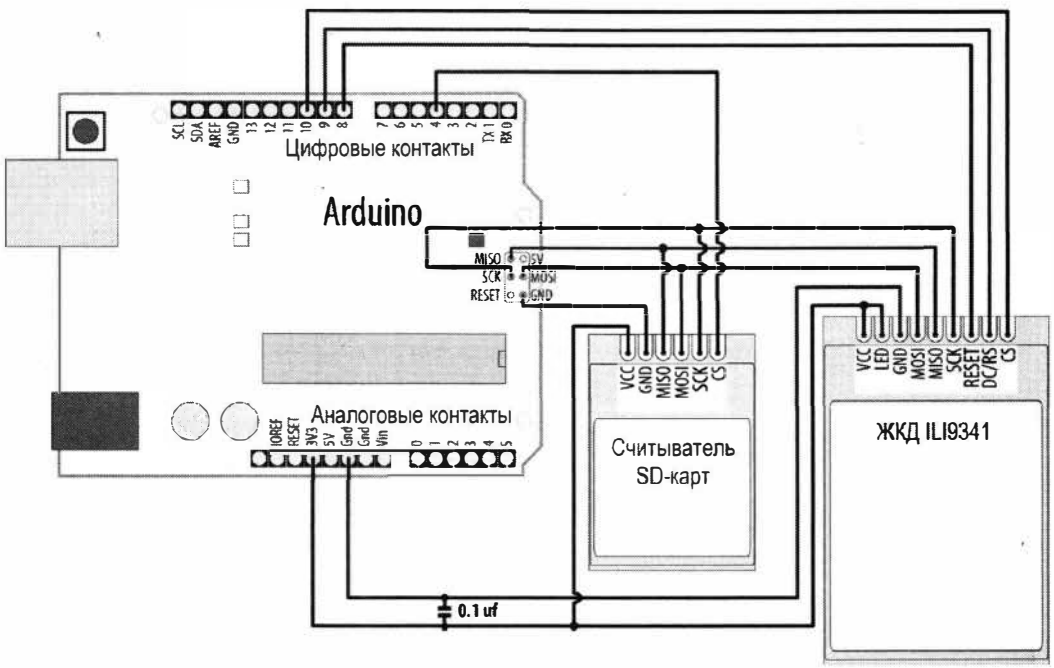


Рис. 13.8. Подключение считывателя SD-карт и ЖК-дисплея к плате Arduino

Обсуждение работы решения и возможных проблем

В решении используются жидкокристаллический TFT-дисплей на основе контроллера ILI9341 и считыватель SD-карт на адаптерной плате, оснащенные интерфейсом SPI. Оба эти устройства предлагаются широким рядом поставщиков. В некоторых случаях TFT-дисплей может быть оснащен встроенным считывателем карт микроSD — как, например, TFT-дисплей компании Adafruit (артикул 1480). При этом для подключения потребуется меньшее количество линий, поскольку TFT-

дисплей и считыватель SD-карт используют общие контакты MISO, MOSI, SCK, GND и VIN.

В скетче решения подключаются несколько библиотек, которые нужно установить с помощью Менеджера библиотек: Adafruit GFX, Adafruit ILI9341 и Adafruit ImageReader. Хотя среда Arduino IDE содержит собственную библиотеку для работы со считывателем SD-карт, библиотека Adafruit ImageReader, которая обеспечивает загрузку изображений с SD-карты, использует модифицированную версию библиотеки SdFat Билла Греймана (Bill Greiman). Эту библиотеку можно найти в списке Менеджера библиотек, выполнив поиск по ключевой фразе: `SdFat - Adafruit Fork`.

Считыватели SD-карт могут быть самых разных видов. Самые простые из них — например, считыватель компании SparkFun (артикул BOB-12941) — представляют собой разъем для SD-карты, впаянный в адаптерную плату. Это возможно, поскольку SD-карты сами могут работать как устройства SPI (провода сигнальных линий интерфейса можно непосредственно подсоединять к контактам SD-карты, и подключать ее таким образом без каких бы то ни было дополнительных промежуточных устройств). Такой тип считывателя может работать только с логическими уровнями с напряжением величиной 3,3 В. А некоторые считыватели SD-карт — например, считыватель компании Adafruit (артикул 254) — содержат переключатель напряжения уровней, что позволяет подавать на них как рабочее питание, так и логические сигналы с напряжением величиной 5 В.

При работе с интерфейсом I²C для выбора определенного устройства используется его уникальный адрес. А в интерфейсе SPI выбор конкретного ведомого устройства ведущим устройством осуществляется подачей сигнала на его контакт CS выбора устройства. В скетче решения линия CS выбора устройства считывателя SD-карт подключена к контакту 4 платы Arduino, а TFT-дисплея — к контакту 10. А линия передачи данных и команд дисплею библиотекой Adafruit ILI9341 подключена к контакту 9.

Скетч создает несколько экземпляров разных объектов: один — для представления SD-карты, другой — для представления корневого каталога или файловой системы карты и еще один — для загрузки изображений с карты и отображения их на TFT-дисплее. В функции `setup()` скетч инициализирует считыватель SD-карт и TFT-дисплей, а затем открывает корневой каталог карты для чтения. В главном цикле `loop()` скетч вызывает функцию `openNext()` для открытия следующего файла, а затем функцию `isBMP()`, чтобы определить, является ли этот файл растровым изображением (файлом с расширением `BMP`). При положительном результате проверки скетч отображает растровое изображение на экране, выдерживает короткую паузу и переходит к обработке следующего файла.

Растровые изображения должны быть сохранены на SD-карте в виде несжатых BMP-файлов с 24-битовой глубиной цвета. Скетч не сможет загружать изображения иного формата.

Дополнительная информация

Дополнительный материал по работе с жидкокристаллическими дисплеями приводится в *разд. 11.9–11.11*.

13.3. Работа с микросхемами на интерфейсе I²C

ЗАДАЧА

Требуется работать с микросхемой, оснащенной интерфейсом I²C, — например, с микросхемой ЭСППЗУ[1] с побитовым доступом. С помощью такой микросхемы можно расширить энергонезависимую память платы Arduino, когда вам нужно получить ее больший объем, чем имеется на плате.

РЕШЕНИЕ

В этом решении используется микросхема ЭСППЗУ 24LC128 с интерфейсом I²C компании Microchip Technology. Подключение микросхемы к плате Arduino показано на рис. 13.9. Если ваша плата Arduino рассчитана на питание 3,3 В, подключите контакт Vcc микросхемы не к контакту 5V, а к контакту 3V3 платы Arduino, чтобы не допустить ее повреждения.

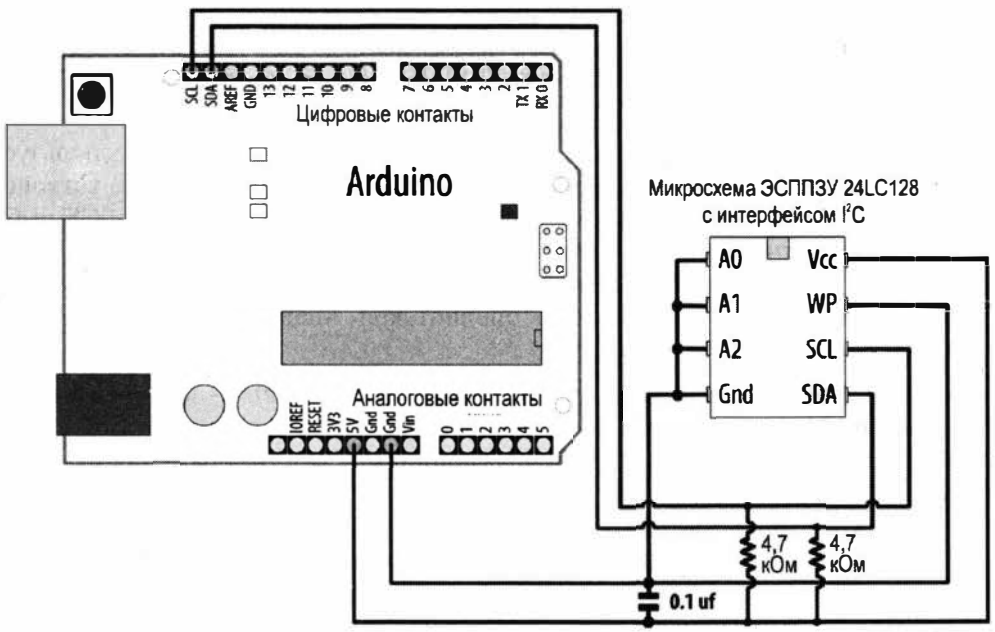


Рис. 13.9. Подключение микросхемы ЭСППЗУ с интерфейсом I2C к плате Arduino

Скетч для работы с подключенной микросхемой приводится в листинге 13.4. Он предоставляет такую же функциональность, что и библиотека EEPROM среды Arduino IDE (см. *разд. 18.1*), но использует внешнюю микросхему ЭСППЗУ, подключенную по интерфейсу I²C, чтобы предоставить пользователю намного больший объем памяти.

---

[1] ЭСППЗУ — электрически стираемое перепрограммируемое ПЗУ (*англ.* EEPROM, Electrically Erasable Programmable Read-Only Memory).

Листинг 13.4. Взаимодействие с микросхемой ЭСППЗУ по интерфейсу I²C

```
/*
* Скетч I2C EEPROM
* Считывает и записывает данные в микросхему памяти 24LC128
*/

#include <Wire.h>

const byte EEPROM_ID = 0x50; // Адрес I2C для микросхемы ЭСППЗУ 24LC128

// Код ASCII первого читаемого символа – 33
int thisByte = 33;

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Требуется для плат Leonardo и большинства плат
                   // с микроконтроллером ARM
   Wire.begin();
   Serial.println("Writing 1024 bytes to EEPROM");
                   Записываем в ЭСППЗУ 1024 байта
   for (int i=0; i < 1024; i++)
   {
      I2CEEPROM_Write(i, thisByte);
      // Переходим к следующему символу
      thisByte++;
      if (thisByte == 126) // Также можно было использовать: if (thisByte == '~')
      thisByte = 33;       // Начинаем сначала
   }
   Serial.println("Reading 1024 bytes from EEPROM");
                   Считываем из ЭСППЗУ 1024 байта
   int thisByte = 33;
   for (int i=0; i < 1024; i++)
   {
      char c = I2CEEPROM_Read(i);
      if(c != thisByte)
      {
         Serial.println("read error"); // Ошибка чтения
         break;
      }
      else
      {
         Serial.print(c);
      }
      thisByte++;
```

```
      if(thisByte == 126)
      {
         Serial.println();
         thisByte = 33; // Начинаем с начала с новой строки
      }
   }
   Serial.println();
   Serial.println("Done."); // Готово
}

void loop()
{
// Здесь ничего не делаем.
}

// Эта функция подобна функции EEPROM.write() среды Arduino IDE
void I2CEEPROM_Write(unsigned int address, byte data)
{
   Wire.beginTransmission(EEPROM_ID);
   Wire.write((int)highByte(address));
   Wire.write((int)lowByte(address));
   Wire.write(data);
   Wire.endTransmission();
   delay(5); // Даем интерфейсу I2C микросхемы ЭСППЗУ завершить цикл записи
}

// Эта функция подобна функции EEPROM.read() среды Arduino IDE
byte I2CEEPROM_Read(unsigned int address )
{
   byte data;
   Wire.beginTransmission(EEPROM_ID);
   Wire.write((int)highByte(address));
   Wire.write((int)lowByte(address));
   Wire.endTransmission();
   Wire.requestFrom(EEPROM_ID,(byte)1);
   while(Wire.available() == 0); // Ожидаем данные

  data = Wire.read();
  return data;
}
```

Обсуждение работы решения и возможных проблем

В этом решении используется микросхема ЭСППЗУ 24LC128 объемом 128 килобит. На рынке предлагаются подобные микросхемы как с большим, так и с меньшим объемом памяти (далее будет приведена ссылка на список таких микросхем компании Microchip). Адрес микросхемы задается установкой соответствующих

уровней на ее контактах A0 по A2 и занимает диапазон значений от 0x50 до 0x57, как показано в табл. 13.2.

Таблица 13.2. *Значения адресов для микросхемы памяти 24LC128*

A0	A1	A2	Адрес
««Земля»»	«Земля»	«Земля»	0x50
+5 В	«Земля»	«Земля»	0x51
«Земля»	+5 В	«Земля»	0x52
+5 В	+5 В	«Земля»	0x53
«Земля»	«Земля»	+5 В	0x54
+5 В	«Земля»	+5 В	0x55
+5 В	+5 В	«Земля»	0x56
+5 В	+5 В	+5 В	0x57

В скетче решения библиотека Wire задействуется так же, как и в других решениях этой главы, поэтому внимательно прочитайте эти решения, чтобы понять код, который инициализирует устройство I²C и запрашивает у него данные.

Операции записи и чтения, специфичные для использованной здесь микросхемы ЭСППЗУ, выполняются функциями: `i2cEEPROM_Write()` и `i2cEEPROM_Read()`. Выполнение этих операций начинается с операции начала передачи:

```
Wire.beginTransmission()
```

по адресу I²C микросхемы памяти. Затем передается 2-байтное значение, определяющее адрес ячейки памяти, по которому должна выполняться операция чтения или записи.

В функции записи за адресом памяти следует значение, которое нужно записать по этому адресу. В нашем случае записывается один байт.

В операции чтения микросхеме ЭСППЗУ посылается адрес ячейки памяти, а затем выполняет функция запроса данных:

```
Wire.requestFrom(EEPROM_ID,(byte)1);
```

которая возвращает один байт из ячейки памяти по указанному адресу.

Операции чтения можно ускорить, заменив 5-секундную паузу проверкой готовности состояния микросхемы ЭСППЗУ записывать новый байт. Дополнительная информация на этот счет приводится в описании метода «Acknowledge Polling» (см. раздел 7 справочного листка микросхемы). Данные также можно записывать страницами объемом 64 байта, а не отдельными байтами. О том, как это делать, подробно рассказано в разделе 6 справочного листка микросхемы.

Микросхема запоминает переданный ей адрес и для каждой следующей операции чтения или записи переходит к следующему адресу. Таким образом, при чтении

нескольких байтов можно задать начальный адрес, а затем выполнить требуемое количество запросов и приемов данных.

Библиотека Wire позволяет считывать или записывать до 32 байтов за один запрос. При попытке чтения или записи большего количества байты будут выбрасываться.

Контакт микросхемы, обозначенный WP, предназначен для установки защиты от записи. Чтобы разрешить запись в микросхему, этот контакт подключается на «землю», как и сделано в решении. А подача на этот контакт напряжения величиной 5 В запрещает запись в микросхему. Эту возможность можно использовать для того, чтобы предотвратить случайное удаление данных, записанных в память для долгосрочного хранения.

Дополнительная информация

Подробная информация по микросхеме ЭСППЗУ 24LC128 приведена в ее справочном листке (**https://oreil.ly/yHXAc**).

Операции чтения можно ускорить, заменив 5-секундную паузу проверкой готовности состояния микросхемы ЭСППЗУ записывать новый байт (см. описание метода «Acknowledge Polling» в разделе 7 справочного листка микросхемы).

Список подобных микросхем ЭСППЗУ на интерфейсе I^2C с различными объемами памяти приводится на веб-странице: **https://oreil.ly/3aLyo**.

Компания Gravitech LLC предлагает шилд датчика температуры для плат Arduino с возможностью сохранения данных в ЭСППЗУ и отображения значения температуры на 4-разрядном 7-сегментном светодиодном дисплее (**https://oreil.ly/fteoP**).

13.4. Увеличение количества портов I^2C

ЗАДАЧА

Требуется подключить к плате Arduino большее количество устройств I^2C, чем это позволяет доступное количество портов на плате.

РЕШЕНИЕ

Эту задачу можно решить с помощью микросхемы расширителя портов I^2C — например, микросхемы PCF8574 или PCF8574A, которая имеет восемь контактов ввода/вывода, которыми можно управлять через интерфейс I^2C. Подключение этой микросхемы к плате Arduino показано на рис. 13.10, а в листинге 13.5 приводится скетч для управления восемью светодиодами через интерфейс I^2C с помощью этой микросхемы.

Если ваша плата Arduino рассчитана на питание 3,3 В, подключите контакт Vcc микросхемы не к контакту 5V, а к контакту 3V3 платы Arduino, чтобы не допустить ее повреждения.

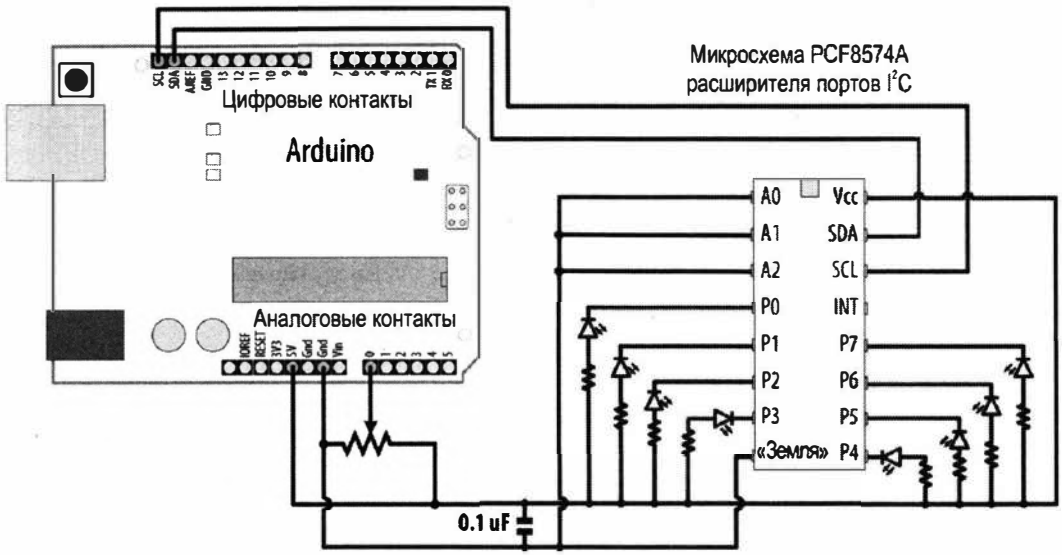


Рис. 13.10. Подключение микросхемы расширителя портов PCF8574A к плате Arduino для управления восемью светодиодами по интерфейсу I²C

Функциональность скетча такая же, как и в скетче из *разд. 7.6*, но для управления светодиодами в нем используется расширитель портов I²C, в результате чего задействуется только два контакта платы Arduino.

Листинг 13.5. Управление восемью светодиодами по интерфейсу I²C с помощью расширителя портов I²C

```
/*
* Скетч I2C bargraph
* Управляет линейным индикатором из восьми светодиодов с помощью интерфейса I2C
* Включает последовательность светодиодов, количество которых
  пропорционально значению, выдаваемому аналоговым датчиком
* См. также решение разд. 7.6
*/

#include <Wire.h>

const int address = 0x20; // Адрес для микросхемы PCF8574.
        // Для микросхемы PCF8574A используйте адрес 0x38
const int NbrLEDs = 8;
const int analogInPin = A0; // Номер аналогового контакта для подключения потенциометра

int sensorValue = 0; // Считываемое с датчика значение
int ledLevel = 0;    // Значение датчика, преобразованное в светодиодный "уровень"
int ledBits = 0;     // Для включения светодиода переменной ledBits
                     // присваивается значение 1
```

```
void setup()
{
   Wire.begin(); // Инициализируем поддержку интерфейса I2C платы Arduino
}

void loop()
{
   sensorValue = analogRead(analogInPin);  // Считываем входной аналоговый сигнал
   ledLevel = map(sensorValue, 0, 1023, 0, NbrLEDs); // Масштабируем
        // полученное значение соответственно количеству светодиодов
   for (int led = 0; led < NbrLEDs; led++)
   {
      Wire.beginTransmission(address);
      if (led < ledLevel)
      {
         Wire.write(~ (1 << led));
      }
      else
      {
         Wire.write(0xFF);     // Выключаем все светодиоды
      }
      Wire.endTransmission(); // Отправляем значение через интерфейс I2C
   }
}
```

Обсуждение работы решения и возможных проблем

Сопротивление токоограничивающих резисторов светодиодов должно быть 220 Ом или больше (подробно о том, как определять величину сопротивления токоограничивающих резисторов, рассказано в *главе 7*).

Посредством функции `analogRead()` скетч считывает текущее значение выходного сигнала потенциометра и масштабирует его к соответствующему значению уровня `ledLevel` в диапазоне от нуля до количества светодиодов. Затем скетч начинает исполнять цикл `for`, который обрабатывает каждый светодиод. Если номер текущего светодиода меньше, чем значение переменной `ledLevel`, скетч включает этот светодиод. Параметр команды `Wire.write()` для активирования контакта микросхемы PCF8574A имеет вид битового поля. Например, значение `0b00000001` (1) устанавливает на контакте 0 высокий уровень, а значение `0b11111111` (255 десятичное) устанавливает высокий уровень на всех контактах. Но для включения светодиода посредством микросхемы PCF8574A на соответствующем контакте микросхемы нужно установить не высокий уровень, а низкий.

Токовые возможности контактов микросхемы PCF8574A более низкие, чем у контактов платы Arduino. В частности, каждый контакт этой микросхемы может предоставлять чрезвычайно малый вытекающий ток — намного меньший, чем требуется для питания светодиода. Но каждый контакт может принимать втекающий ток величиной до 25 мА. Это означает, что для управления светодиодами посредством

микросхемы PCF8574A необходимо использовать обратную логику, подобную логике считывания состояния кнопок при задействовании внутренних повышающих резисторов посредством оператора `INPUT_PULLUP` (см. *раз. 2.4*). Это и есть та причина, по которой управляемые светодиоды подключены своим вторым выводом к шине питания +5 В (или +3,3 В), а не к шине «земли». Когда на контакте микросхемы PCF8574A устанавливается низкий уровень, ток поставляется из шины положительного напряжения и протекает на «землю» через этот контакт. Этим объясняется использование в скетче булевого оператора ~ (НЕ) — для инвертирования значения. Таким образом, двоичное значение `0b00000001` (или 1 в десятичной системе) становится `0b11111110` (или 254 в десятичной системе).

Но это еще не все токовые ограничения микросхемы PCF8574A — она не может принимать одновременно более чем 80 мА входящего тока. Впрочем, если одновременно включить все светодиоды (значение `0b00000000`), микросхема, скорее всего, смогла бы справиться с этим, но поскольку такой ток сильно превышает ее возможности, срок ее службы будет сокращен. По этой причине скетч одновременно включает только один светодиод, используя для этого булев сдвиг влево для вычисления бита, который нужно задействовать, а затем инвертируя его. Например, контакт 0 включается значением `0b11111110`, а контакт 3 — значением `0b11110111`. Однако эта операция выполняется настолько быстро, что человеческий глаз не успевает уловить ее, и поэтому пользователю кажется, что все диоды включены одновременно. Хотя вызывать функцию `Wire.write(0xFF);` для невключенного светодиода совсем не обязательно, вызов ее обеспечивает то, что скетч всегда выполняет одинаковое количество команд. Это удерживает постоянный уровень яркости светодиодов, независимо от того, сколько из них включено.

Если наблюдается мерцание светодиодов, его можно минимизировать, оставаясь в пределах токовых возможностей микросхемы PCF8574A, — включая одновременно четыре светодиода. Соответствующий фрагмент кода приводится в листинге 13.6.

Листинг 13.6. Минимизирование мерцания светодиодов включением одновременно четырех светодиодов

```
int bitField = 0;
for (int led = 0; led < NbrLEDs; led++)
{
   if (led < ledLevel)
   {
      bitField |= (1 << led);
   }
   if ((led + 1) % 4 == 0) // Отправляем команду на каждые четыре контакта
   {
      Wire.beginTransmission(address);
      Wire.write(~bitField);
      Wire.endTransmission(); // Отправляем значение через интерфейс I2C
      bitField = 0; // Очищаем битовое поле
   }
}
```

Адрес микросхемы можно изменить, поменяв комбинацию уровней на ее контактах A0, A1 и A2, как показано в табл. 13.3. Для микросхемы PCF8574A, смонтированной на адаптерной плате, это можно сделать с помощью проволочных или пропаянных перемычек.

Таблица 13.3. *Значения адресов для микросхемы памяти PCF8574A*

A0	A1	A2	Адрес PCF8574A	Адрес PCF8574
«Земля»	«Земля»	«Земля»	0x38	0x20
+5 В	«Земля»	«Земля»	0x39	0x21
«Земля»	+5 В	«Земля»	0x3A	0x22
+5 В	+5 В	«Земля»	0x3B	0x23
«Земля»	«Земля»	+5 В	0x3C	0x24
+5 В	«Земля»	+5V	0x3D	0x25
+5 В	+5 В	«Земля»	0x3E	0x26
+5 В	+5 В	+5 В	0x3F	0x27

При использовании микросхемы расширителя портов для ввода считывание байта с микросхемы можно выполнять с помощью следующего кода:

```
Wire.requestFrom(address, 1);
if(Wire.available())
{
   data = Wire.receive();
   Serial.println(data,BIN);
}
```

Дополнительная информация

Подробная информация по микросхеме расширителя портов I^2C PCF8574A приводится в ее справочном листке (**https://oreil.ly/WjEFj**).

В *разд. 13.1* рассматривается решение, которое может работать с более сильными токами.

13.5. Организация взаимодействия нескольких плат Arduino

ЗАДАЧА

Требуется организовать взаимодействие двух или более плат Arduino. Например, вам нужны большие возможности ввода/вывода или большая вычислительная мощность, чем может быть обеспечена одной платой.

РЕШЕНИЕ

Эту задачу можно решить, используя для обмена данными между платами интерфейс I²C. Подключение плат Arduino друг к другу показано на рис. 13.11, а обмен данными между ними демонстрируется в скетчах из листингов 13.7 и 13.8.

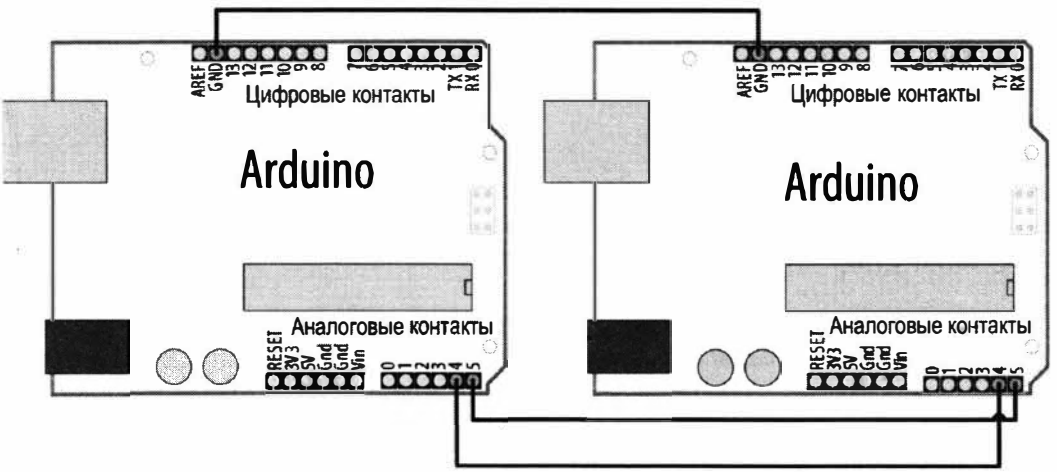


Рис. 13.11. Подключение двух плат Arduino по протоколу I²C: одна плата — ведущая, а другая — ведомая

Скетч из листинга 13.7 демонстрирует передачу ведущей платой ведомой плате через интерфейс I²C символов, полученных по последовательному интерфейсу с компьютера.

Листинг 13.7. Передача символов ведущей платой ведомой плате

```
/*
* Скетч I2C Master
* Принимает данные с компьютера по последовательному порту и передает
  их ведомой плате по интерфейсу I2C
*/

#include <Wire.h>

const int address = 4; // Адрес ведомой платы

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

void loop()
{
   char c;
```

```
   if(Serial.available() > 0)
   {
      c = Serial.read();
      // send the data
      Wire.beginTransmission(address); // Передаем полученные данные
                                       // ведомой плате
      Wire.write(c);
      Wire.endTransmission();
   }
}
```

А скетч из листинга 13.8 исполняется на ведомой плате Arduino и передает данные, полученные по интерфейсу I^2C от ведущей платы, на компьютер по последовательному интерфейсу.

Листинг 13.8. Прием данных по интерфейсу I^2C и передача их по последовательному интерфейсу

```
/*
* Скетч I2C Secondary
* Отслеживает запросы по интерфейсу I2C и перенаправляет их на последовательный порт
*/

#include <Wire.h>

const int address = 4; // Адрес, используемый взаимодействующими устройствами

void setup()
{
   Serial.begin(9600);
   Wire.begin(address);          // Подключаемся к шине I2C, используя этот адрес
   Wire.onReceive(receiveEvent); // Регистрируем событие для обработки запросов
}

void loop()
{
   // Здесь ничего не делаем — вся работа выполняется функцией receiveEvent()
}

void receiveEvent(int howMany)
{
   while(Wire.available() > 0)
   {
      char c = Wire.read(); // Принимаем байт в виде символа
      Serial.write(c); // Перенаправляем на последовательный порт
   }
}
```

Обсуждение работы решения и возможных проблем

В этой главе основное внимание уделяется использованию платы Arduino в качестве ведущего устройства I²C, обращающегося к различным ведомым устройствам I²C. Здесь же ведомым устройством является другая плата Arduino, которая отвечает на запросы ведущей платы. В этом решении можно использовать методы для передачи данных, рассмотренные в *главе 4*. Например, отправлять данные с помощью метода (функции) `print()`.

Так, скетч из листинга 13.9 отправляет выходные данные по интерфейсу I²C с помощью функции `Wire.println()`. Используя этот скетч совместно со скетчем для ведомого устройства (см. листинг 13.8), можно выводить на печать данные от ведущего устройства, не прибегая к использованию последовательного порта (последовательный порт ведомого устройства задействуется для отображения выходных данных).

Листинг 13.9. Скетч для отправки данных с помощью функции `Wire.println()`

```
/*
* Скетч I2C Master w/print
* Отправляет данные с датчика по интерфейсу I2C на ведомую плату
  Arduino, используя функцию print()
*/

#include <Wire.h>

const int address = 4;     // Адрес, используемый взаимодействующими устройствами
const int sensorPin = A0; // Номер аналогового контакта для подключения потенциометра

int val; // Переменная для хранения значения, полученного с датчика (потенциометра)

void setup()
{
   Wire.begin();
}

void loop()
{
   val = analogRead(sensorPin);      // Считываем напряжение на контакте потенциометра
   // (val ranges from 0 to 1023)
   Wire.beginTransmission(address); // Передаем полученные данные ведомой плате
   Wire.println(val);
   Wire.endTransmission();
   delay(1000);
}
```

Скетч из листинга 13.10 обрабатывает несколько значений (см. решение из *разд. 4.5*), но использует не последовательный интерфейс, а интерфейс I²C. Он от-

правляет значения с первых трех аналоговых контактов в текстовом сообщении в формате: H3, v0, v1, v2. Буква H в первом значении указывает на начало собственно сообщения (заголовок), цифра 3 означает, что сообщение будет содержать три значения, а идентификаторы v0, v1, v2 представляют числовые значения сигналов, считываемые с трех аналоговых вводов.

Листинг 13.10. Обработка нескольких аналоговых значений с использованием интерфейса I^2C

```
/*
* Скетч I2C Master multiple
* Отправляет значения с нескольких датчиков на ведомую плату Arduino
  по интерфейсу I2C, используя функцию print()
*/

#include <Wire.h>

const int address = 4;  // Адрес, используемый взаимодействующими
                        // устройствами
const int firstSensorPin = A0; // Номер первого из трех
                               // последовательных аналоговых контактов
const int nbrSensors = 3;      // Будут использоваться три
                               // последовательных аналоговых контакта

int val;                       // Переменная для хранения значения,
                               // полученного с датчика

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

void loop()
{
   Wire.beginTransmission(address); // Передаем полученные данные ведомой плате
   Wire.print('H'); // Односимвольный заголовок, обозначающий начало сообщения
   Wire.print(nbrSensors);
   for (int i = 0; i < nbrSensors; i++)
   {
      val = analogRead(firstSensorPin + 1); // Считываем значение сигнала датчика
      Wire.print(','); // Символ разделителя (запятая)
      Wire.print(val);
   }
   Wire.println(); // Конец сообщения
   Wire.endTransmission();
   delay(100);
}
```

А скетч из листинга 13.11 исполняется на ведомой плате Arduino и принимает сообщения, передаваемые ему скетчем из листинга 13.10, с последующим выводом их в окно монитора порта.

Листинг 13.11. Прием сообщений с несколькими значениями и отображение их в окне монитора порта

```
/*
* Скетч I2C Secondary multiple
* Отслеживает запросы по интерфейсу I2C и перенаправляет их на последовательный порт
*/

#include <Wire.h>

const int address = 4; // Адрес, используемый взаимодействующими устройствами

void setup()
{
   Serial.begin(9600);
   Wire.begin(address);          // Подключаемся к шине I2C, используя этот адрес
   Wire.onReceive(receiveEvent); // Регистрируем событие для обработки запросов
}

void loop()
{
   // Здесь ничего не делаем – вся работа выполняется
   // функцией receiveEvent()
}

void receiveEvent(int howMany)
{
   while(Wire.available() > 0)
   {
      char c = Wire.read(); // Принимаем байт в виде символа
      if( c == 'H')
      {
         // Переходим сюда, если начало сообщения
         int nbrSensors = Wire.parseInt();
         if(nbrSensors > 0)
         {
            for(int i=0; i < nbrSensors; i++ )
            {
               int val = Wire.parseInt();
               Serial.print(val); Serial.print(" ");
            }
            Serial.println();
         }
      }
   }
}
```

Дополнительная информация

Дополнительная информация по функции `print()` среды Arduino IDE приводится в *главе 4*.

13.6. Использование возможностей акселерометра контроллера Wii Nunchuk

ЗАДАЧА

Вы хотите подключить к плате Arduino модуль игрового контроллера Nunchuk консоли Wii (Wii Nunchuk), чтобы воспользоваться его возможностями акселерометра. Wii Nunchuk — популярный недорогой игровой гаджет, с помощью которого можно определять ориентацию устройства в пространстве, измеряя силу земного притяжения. Вы можете использовать как оригинальный контроллер Wii Nunchuk, так и его клоны, предлагаемые сторонними производителями. Второй вариант обычно намного дешевле.

РЕШЕНИЕ

Контроллер Wii Nunchuk обычно подключается к консоли Wii через собственный фирменный разъем. Если вы не планируете использовать контроллер с консолью Wii, то этот разъем можно просто отрезать и припаять к проводам штыревые разъемы. Или же, чтобы не портить гаджет, можно взять небольшую перфоплату и вывести на нее контакты разъема, схема расположения которых показана на рис. 13.12.

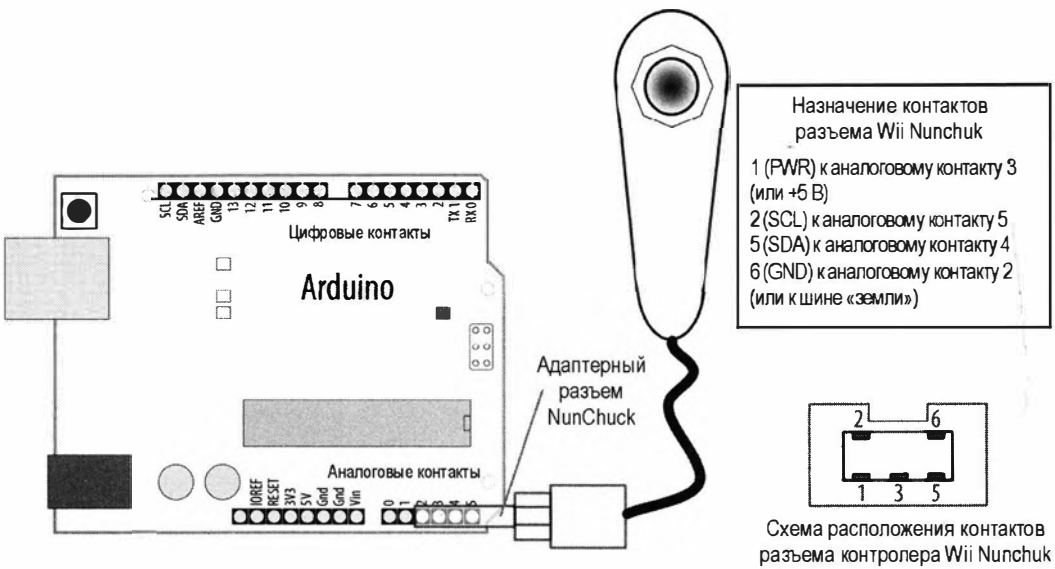


Рис. 13.12. Подключение контроллера Wii Nunchuk к плате Arduino

Можно также приобрести адаптерную плату NunChucky Wii Nunchuck I²C Breakout Adapter производства компании Solarbotics (артикул 31040, **https://solarbotics.com/product/31040/**).

Адаптерные разъемы — такие как NunChucky, предполагают работу с платой Arduino, в которой сигнал SDA доступен на контакте A4, а сигнал SCL — на контакте A5. Такая разводка этих сигналов соответствует платам Arduino Uno и большинству плат с микроконтроллером ATmega328 и с более ранними микроконтроллерами типа ATmega168. Но для плат с микроконтроллером ARM (например, Arduino Leonardo и многих других) такая разводка этих сигналов недоступна. Поэтому при работе с одной из таких плат или с платой, имеющей иную организацию контактов, чем плата Arduino Uno, контакты сигналов SDA и SCL адаптера следует подключить к соответствующим контактам используемой платы, а контакты питания адаптера подключить к контактам 3V3 и GND платы Arduino. При этом можно удалить из скетча строки кода

```
digitalWrite(gndPin, LOW);
```

и

```
digitalWrite(vccPin, HIGH);
```

В листинге 13.12 приводится скетч Arduino, который отправляет данные с контроллера Wii Nunchuk скетчу Processing.

Листинг 13.12. Отправка данных с контроллера Wii Nunchuk скетчу Processing

```
/*
* Скетч nunchuck_lines
* Принимает данные с контроллера Wii Nunchuk и отправляет их скетчу
  Processing, который прорисовывает линию, наклон которой
  соответствует наклону контроллера
*/

#include <Wire.h>          // Подключаем библиотеку Wire

const int vccPin = A3;     // Плюс питания контроллера берется с контакта 17
const int gndPin = A2;     // Минус питания ("земля") контроллера берется
                           // с контакта 16
const int dataLength = 6; // Количество байтов в запросе

static byte rawData[dataLength]; // Массив для хранения данных,
                                 // полученных с контроллера

enum nunchuckItems { joyX, joyY, accelX, accelY, accelZ, btnZ, btnC };

void setup()
{
   pinMode(gndPin, OUTPUT); // Задаем правильное состояние для контактов питания
   pinMode(vccPin, OUTPUT);
   digitalWrite(gndPin, LOW);
```

```
   digitalWrite(vccPin, HIGH);
   delay(100); // Пауза, чтобы дать время всему стабилизироваться
   Serial.begin(9600);
   nunchuckInit();
}

void loop()
{
   nunchuckRead();
   int acceleration = getValue(accelX);
   if((acceleration >= 75) && (acceleration <= 185))
   {
      // Масштабирует диапазон значений от 75 до 185
      // к диапазону значений от 0 до 63
      byte x = map(acceleration, 75, 185, 0, 63);
      Serial.write(x);
      delay(20); // Время в миллисекундах между прорисовками
   }
}

void nunchuckInit()
{
   Wire.begin(); // Подключаемся к шине I2C как ведущее устройство
   Wire.beginTransmission(0x52); // Передаем на устройство 0x52
   Wire.write((byte)0x40);       // Передаем адрес памяти
   Wire.write((byte)0x00);       // Передаем значение 0
   Wire.endTransmission();       // Завершаем передачу
}

// Отправляет контроллеру Wii Nunchuk запрос данных
static void nunchuckRequest()
{
   Wire.beginTransmission(0x52); // Передаем на устройство 0x52
   Wire.write((byte)0x00);       // Передаем один байт
   Wire.endTransmission();       // завершаем передачу
}

// Принимает данные с контроллера Wii Nunchuk
// Возвращает true при успешном приеме, false - в противном случае
bool nunchuckRead()
{
   int cnt=0;
   Wire.requestFrom (0x52, dataLength); // Запрос данных с контроллера Wii Nunchuk
   while (Wire.available ())
   {
      rawData[cnt] = nunchuckDecode(Wire.read());
      cnt++;
   }
```

```
    nunchuckRequest();    // Передаем запрос следующего пакета данных
    if (cnt >= dataLength)
       return true;       // Успешное завершение, если получены все 6 байтов
    else
    return false; //Неудача
}

// Кодируем данные в формат, принимаемый большинством
// драйверов Wii Remote
static char nunchuckDecode (byte x)
{
    return (x ^ 0x17) + 0x17;
}

int getValue(int item)
{
    if (item <= accelZ)
    return (int)rawData[item];
    else if (item == btnZ)
    return bitRead(rawData[5], 0) ? 0: 1;
    else if (item == btnC)
    return bitRead(rawData[5], 1) ? 0: 1;
}
```

Обсуждение работы решения и возможных проблем

Протокол I²C часто используется в коммерческих продуктах — таких как контроллер Wii Nunchuk — в целях организации связи между устройствами. Для контроллера Wii Nunchuk не предлагается официального справочного листка, но его сигналы были проанализированы (подвержены обратному инжинирингу), чтобы определить команды, требуемые для взаимодействия с ним.

В листинге 13.13 приводится скетч Processing, который принимает данные с контроллера Wii Nunchuk, передаваемые ему скетчем Arduino (см. листинг 13.12), и отображает в окне приложения линию, наклон которой соответствует наклону контроллера, как показано на рис. 13.13 (дополнительные сведения о приеме данных с Arduino по последовательному интерфейсу с помощью средств Processing

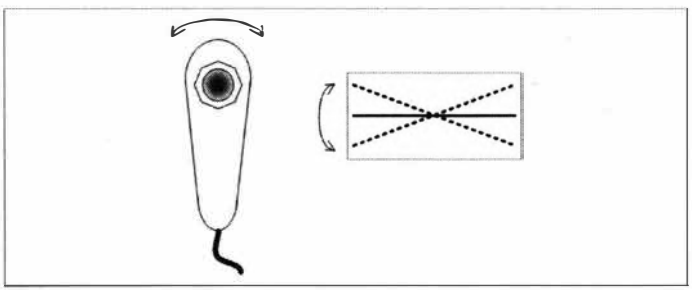


Рис. 13.13. Данные о перемещении контроллера Wii Nunchuk, отображенные в виде наклонных линий

приводятся в *главе 4*, там же дается информация по организации использования языка Processing применительно с Arduino).

Листинг 13.13. Прием данных от контроллера Wii Nunchuk с платы Arduino и отображение их в виде линий

```
// Скетч Processing для рисования линий, наклон которых соответствует
   наклону контроллера
import processing.serial.*;

Serial myPort; // Создаем экземпляр объекта класса Serial

public static final short portIndex = 1;

void setup()
{
   size(200, 200);
   // Открываем порт, используемый для Bluetooth радио -(см. главу 4)
   myPort = new Serial(this,Serial.list()[portIndex], 9600);
}

void draw()
{
   if ( myPort.available() > 0)
   {
      // Если присутствуют данные,
      int y = myPort.read(); // считываем их и сохраняем в переменной val
      background(255);       // Задаем белый цвет фона
      line(0,63-y,127,y);    // рисуем линию
   }
}
```

Скетч Arduino в листинге 13.12 подключает библиотеку Wire среды Arduino IDE и определяет номера контактов платы Arduino, используемые для подачи питания на контроллер Wii Nunchuk:

```
#include <Wire.h>       // Подключаем библиотеку Wire

const int vccPin = A3; // Плюс питания контроллера берется с контакта 17
const int gndPin = A2; // Минус питания ("земля") контроллера берется
                       // с контакта 16
```

Библиотека Wire предназначена для работы с интерфейсом I^2C и входит в состав среды Arduino IDE. Идентификатор A3 обозначает аналоговый контакт 3 (цифровой контакт 17), а идентификатор A2 — аналоговый контакт 2 (цифровой контакт 16). Эти контакты используются для подачи питания на контроллер Wii Nunchuk.

Следующая далее строка кода:

```
enum nunchuckItems { joyX, joyY, accelX, accelY, accelZ, btnZ, btnC };
```

представляет собой структуру перечисления для создания списка констант. В нашем случае это список значений датчиков, возвращаемых контроллером Wii Nunchuk. Эти константы служат для идентификации запросов значения одного из датчиков контроллера.

Функция `setup()` инициализирует контакты, используемые для подачи питания на контроллер, устанавливая высокий уровень (`HIGH`) на контакте `vccPin` и низкий уровень (`LOW`) на контакте `gndPin`:

```
digitalWrite(gndPin, LOW);
digitalWrite(vccPin, HIGH);
```

Этот код требуется только в том случае, если питание контроллера предоставляется адаптерным разъемом. Использовать цифровые контакты в качестве источника питания обычно не рекомендуется, за исключением тех случаев, когда можно быть уверенным (как в случае с контроллером Wii Nunchuk), что потребление тока питаемым таким образом устройством не превышает максимальную токовую возможность контакта платы Arduino (которая составляет 40 мА — см. *главу 5*).

Функция `nunchuckInit()` устанавливает связь по интерфейсу I²C с контроллером Wii Nunchuk, начиная с вызова функции `Wire.begin()`. В нашем примере плата Arduino, будучи ведущим устройством, отвечает за инициализацию требуемого ведомого устройства — контроллера Wii Nunchuk, находящегося по I²C адресу `0x52`.

Следующая строка кода дает указание библиотеке Wire подготовиться к отправке сообщения устройству по шестнадцатеричному адресу 52 (`0x52`):

```
beginTransmission(0x52);
```

В документации на интерфейс I²C адреса обычно указываются шестнадцатеричными значениями, поэтому вполне целесообразно использовать этот формат и в скетчах.

Функция `Wire.write()` помещает данные значения в буфер библиотеки Wire, где они хранятся до вызова функции `Wire.endTransmission()`, которая, собственно, и осуществляет отправку данных.

Функции `nunchuckRequest()` и `nunchuckRead()` соответственно запрашивают и считывают данные с контроллера Wii Nunchuk.

Функция `Wire.requestFrom()`получает шесть байтов данных с устройства по адресу `0x52` (контроллер Wii Nunchuk).

Контроллер Wii Nunchuk возвращает запрошенные данные в шести байтах, как расписано в табл. 13.4.

Функция `Wire.available()` работает подобно функции `Serial.available()` (см. *главу 4*), указывая количество полученных байтов, но для интерфейса I²C, а не для последовательного интерфейса. При наличии данных они считываются посредством функции `Wire.read()`, а затем декодируются с помощью функции `nunchuckDecode()`. Декодирование данных требуется, чтобы преобразовать полученные значения в числа, понятные для скетча, которые затем сохраняются в буфере `rawData`. Затем отправля-

***Таблица 13.4.** Значения шести байтов данных, возвращаемых контроллером Wii Nunchuk*

Номер байта	Описания
Байт 1	Аналоговое значение оси X джойстика
Байт 2	Аналоговое значение оси Y джойстика
Байт 3	Значение ускорения по оси X
Байт 4	Значение ускорения по оси Y
Байт 5	Значение ускорения по оси Z
Байт 6	Состояния кнопок и самые младшие биты значений ускорения

ется запрос следующих шести байтов данных, чтобы они были готовыми к следующему вызову для получения данных:

```
int acceleration = getValue(accelX);
```

Функции `getValue()` передается одна из переменных из списка датчиков — здесь это переменная ускорения по оси X `accelX`.

Скетчу Processing можно отправлять дополнительные значения, разделяя их запятыми (см. *разд. 4.4*). В листинге 13.14 приводится модифицированный код главного цикла `loop()` для реализации этого.

Листинг 13.14. Оправка скетчу Processing нескольких значений

```
void loop()
{
   nunchuckRead();
   Serial.print("H,"); // Заголовок
   for(int i=0; i < 3; i++)
   {
      Serial.print(getValue(accelX+ i), DEC);
      if( i > 2)
         Serial.write(',');
      else
         Serial.write('\n') ;
   }
   delay(20); // Время в миллисекундах между прорисовками
}
```

Дополнительная информация

Дополнительная информация о библиотеке для взаимодействия с контроллером Wii Nunchuk приводится в *разд. 16.5*.

В *разд. «Обсуждение работы решения и возможных проблем» разд. 4.4* рассматривается скетч Processing, который отображает в режиме реального времени полосовую диаграмму значений акселерометра.

ГЛАВА 14

Простая беспроводная связь

14.0. Введение

Платформа Arduino обладает прекрасными возможностями взаимодействия с окружающей средой, но иногда вам может понадобиться организовать общение с платой Arduino на расстоянии, без ограничений проводного подключения и без накладных расходов вычислительной мощности на реализацию полномасштабного подключения по стеку протоколов TCP/IP. В этой главе мы рассмотрим простые модули беспроводной связи для приложений, в которых главным требованием является низкая стоимость, а также универсальные модули беспроводной связи XBee и Bluetooth, обладающие более продвинутыми возможностями.

Реализовать надежную и безопасную связь между устройствами позволяют такие простые радиомодули пакетной связи, как, например, модуль RFM69HCW. Радиомодули XBee обеспечивают плату Arduino более гибкими возможностями радиосвязи, но эта гибкость может иногда сбивать с толку. В этой главе представлены примеры использования таких модулей: от простой замены беспроводным подключением связи по последовательному порту до ячеистой сети для подключения множества датчиков к нескольким платам Arduino.

Технологии Bluetooth Classic и Bluetooth Low Energy — это популярные варианты взаимодействия плат Arduino с компьютерами и мобильными телефонами. Поскольку в наши дни эти устройства, как правило, оснащены модулями Bluetooth, технологии Bluetooth предлагают удобные способы организации беспроводных соединений с ними без необходимости использования на них какого-либо дополнительного специального оборудования.

14.1. Обмен сообщениями с помощью недорогих радиомодулей

ЗАДАЧА

Требуется организовать обмен сообщениями между двумя платами Arduino, используя для этого недорогое оборудование.

РЕШЕНИЕ

Эта задача решается с помощью простых модулей приемопередатчика RFM69HCW, работающих в нелицензируемом диапазоне радиочастот ISM (Industrial, Scientific,

Medical — промышленный, научный, медицинский). В зависимости от региона, в котором планируется использование таких модулей, необходимо выбрать модуль с соответствующим частотным диапазоном. Модули с частотным диапазоном 433 МГц предназначены для использования в странах региона 1: Европа, Африка, страны бывшего СССР, Монголия и страны Ближнего Востока, лежащие к западу от Персидского залива. Такие модули, смонтированные на адаптерных платах, предлагаются, например, компаниями SparkFun (артикул WRL-12823) и Adafruit (артикул 3071). Модули с частотным диапазоном 915 МГц предназначены для использования в странах региона 2: Северная и Южная Америка, Гренландия и часть восточных островов Тихого океана. Такие модули, смонтированные на адаптерных платах, также предлагаются компаниями SparkFun (артикул WRL-12775) и Adafruit (артикул 3070).

На рынке предлагаются и модули, не установленные на адаптерную плату, но существующие адаптерные платы содержат схемы, облегчающие их подключение. В частности, адаптерные платы компании Adafruit включают преобразователи логических уровней, что позволяет устанавливать на них радиомодули с логическими уровнями с напряжением как 3,3 В, так и 5 В (но в случае использования платы Arduino с рабочим напряжением 3,3 В контакт VIN модуля необходимо подключить к контакту 3V3 платы Arduino, а не к контакту 5V). Модуль необходимо настроить на работу на соответствующей частоте, а затем подключить его к платам Arduino, как показано на рис. 14.1.

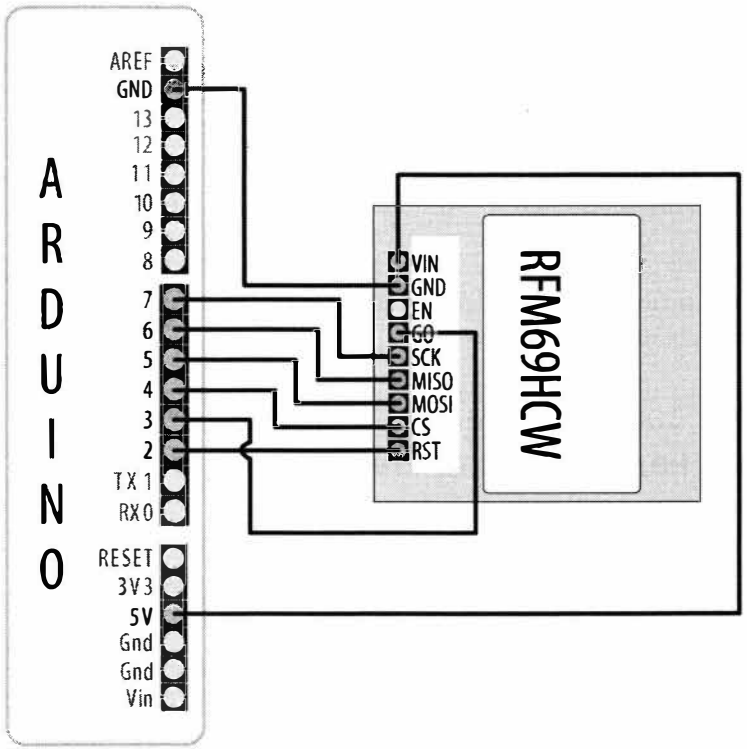


Рис. 14.1. Подключение радиомодуля RFM69HCW к плате Arduino

Для работы с радиомодулями RFM69HCW требуются два скетча: один для передачи (листинг 14.1), а другой для приема (листинг 14.2). Скетч передатчика отправляет короткое сообщение скетчу приемника, который перенаправляет полученное сообщение на монитор порта и посылает ответ.

В обоих скетчах используется библиотека RadioHead от Майка МакКоли (Mike McCauley), которая предоставляет обобщенный интерфейс для разных типов радиоустройств. Библиотеку можно загрузить на ее веб-сайте (**https://oreil.ly/3kZca**), но и компания Adafruit, и компания SparkFun предлагают свои фирменные версии библиотеки, которые обеспечивают несколько лучшую поддержку их устройствам. Загрузите и установите обе эти библиотеки. Библиотеку Adafruit можно загрузить здесь: **https://oreil.ly/cRcdE**, а библиотеку SparkFun — **https://oreil.ly/UmESI** (подробно об установке библиотек рассказано в *разд. 16.2*). Однако в случае использования универсального радиомодуля или какого-либо другого радиомодуля, поддерживаемого библиотекой RadioHead, следует использовать оригинальную версию этой библиотеки, если только производитель или поставщик используемого модуля не предоставляет версию библиотеки, специализированную под свой модуль.

Листинг 14.1. Скетч передатчика

```
/*
* Скетч RFM69HCW transmit
* Отправляет сообщение другому модулю и ожидает ответ
*/

#include <SPI.h>
#include <RH_RF69.h>
#include <RHReliableDatagram.h>

#define MY_ADDR 2   // Адрес этого узла
#define DEST_ADDR 1 // Адрес другого узла

#define RF69_FREQ 915.0 // Устанавливаем поддерживаемую частоту

// Определяем объект радиомодуля
#define RFM69_INT 3
#define RFM69_CS 4
#define RFM69_RST 2
RH_RF69 rf69(RFM69_CS, RFM69_INT);

// Этот объект управляет доставкой сообщения
RHReliableDatagram rf69_manager(rf69, MY_ADDR);

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

```
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(RFM69_RST, OUTPUT);
   digitalWrite(RFM69_RST, LOW);

   Serial.println("Resetting radio"); // Сбрасываем настройки радиомодуля
   digitalWrite(RFM69_RST, HIGH); delay(10);
   digitalWrite(RFM69_RST, LOW); delay(10);
   if (!rf69_manager.init())
   {
      Serial.println("Could not start the radio");
                     // Не удалось запустить радиомодуль
      while (1); // Останов
   }

   if (!rf69.setFrequency(RF69_FREQ))
   {
      Serial.println("Could not set frequency");
                     // Не удалось установить частоту
      while (1); // Останов
   }

   // В случае работы с версией радиомодуля RF69 с большой
   // мощностью (RFM69HW/HCW) требуется использовать следующий код:
   rf69.setTxPower(20, true); // Диапазон мощностью от 14 до 20
   // Каждый модуль должен использовать один и тот же ключ
   uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                     0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
   rf69.setEncryptionKey(key);

   Serial.print("RFM69 radio running at "); // Модуль RFM69 работает на частоте:
   Serial.print((int)RF69_FREQ); Serial.println(" MHz");
}

byte response[RH_RF69_MAX_MESSAGE_LEN]; // Содержит сообщение от другого радиомодуля
byte message[] = "Hello!";

void loop()
{
   delay(1000); // Пауза длительностью в 1 секунду

   if (rf69_manager.sendtoWait((byte *)message, strlen(message), DEST_ADDR))
   {
      byte len = sizeof(response);
      byte sender; // Номер ID отправителя

      // Ждем ответа
      if (rf69_manager.recvfromAckTimeout(response, &len, 2000, &sender))
```

```
    {
      response[len] = 0; // Добавляем null (0) в конец ответа

      // Получили
      Serial.print("Got ["); Serial.print((char *) response);
      Serial.print("] from "); Serial.println(sender);        // от

      // Мигаем светодиодом
      digitalWrite(LED_BUILTIN, HIGH); delay(250);
      digitalWrite(LED_BUILTIN, LOW); delay(250);
    }
    else
    {
      Serial.print("Received no reply from "); Serial.println(sender);
                  // Не получен ответ от
    }
  }
  else
  {
    Serial.print("Failed to send message to "); Serial.println(DEST_ADDR);
                //Не удалось отправить сообщение
  }
}
```

Скетч приемника почти такой же, как и скетч передатчика, но с двумя различиями: первое — не нужно определять адрес назначения `DEST_ADDR`, и второе — значение определения `MY_ADDR` надо заменить на `1` (листинг 14.2).

Листинг 14.2. Фрагмент скетча приемника (откорректированные определения)

```
/*
* Скетч RFM69HCW receive
* Принимает сообщение от первого модуля и отправляет ответ
*/

#include <SPI.h>
#include <RH_RF69.h>
#include <RHReliableDatagram.h>

#define MY_ADDR 1 // Адрес этого узла
```

Затем нужно заменить главный цикл `loop()` и два определения перед ним кодом из листинга 14.3.

Листинг 14.3. Фрагмент скетча приемника (новый цикл `loop()`)

```
byte message[RH_RF69_MAX_MESSAGE_LEN]; // Содержит сообщение от другого радиомодуля
byte reply[] = "Goodbye!";
```

```
void loop()
{
   if (rf69_manager.available()) // Получили сообщение
   {
      byte len = sizeof(message);
      byte sender; // Номер ID отправителя
      if (rf69_manager.recvfromAck(message, &len, &sender)) // Ожидаем сообщение
      {
         message[len] = 0; // Добавляем null (0) в конец сообщения

         Serial.print("Got ["); Serial.print((char *) message);
         Serial.print("] from "); Serial.println(sender);

         // Blink the LED
         digitalWrite(LED_BUILTIN, HIGH); delay(250);
         digitalWrite(LED_BUILTIN, LOW); delay(250);

         // Reply to sender
         if (!rf69_manager.sendtoWait(reply, sizeof(reply), sender))
         {
            Serial.print("Failed to send message to ");
                        // Не удалось отправить сообщение
            Serial.println(sender);
         }
      }
   }
}
```

Обсуждение работы решения и возможных проблем

Библиотека RadioHead содержит несколько драйверов, поддерживающих широкий круг радиоустройств. Чтобы использовать эту библиотеку, необходимо определить драйвер радиомодуля (например, RH_RF69). Затем в функции `setup()` выполняется последовательность операций инициализации. Точный формат определения и инициализации зависит от используемого радиомодуля, но по завершении этих начальных операций остальной код по большому счету похож для всех устройств.

В начале скетча (см. листинг 14.1) подключаются две версии библиотеки RadioHead и библиотека SPI. Первая версия библиотеки RadioHead (RF_RF69) предназначена для работы с линейкой радиомодулей RFM69, а вторая (RHReliableDatagram) является интерфейсом API для надежной доставки сообщения конкретному модулю (узлу) радиосети.

Затем осуществляется определение адреса узла `MY_ADDR`: 2 — для модуля передатчика (см. листинг 14.1) и 1 — для модуля приемника (см. листинг 14.2). В скетче передатчика также определяется адрес узла приемника `DEST_ADDR` (1). Далее задается рабочая частота сети: 915.0 или 433.0 МГц — в зависимости от региона ISM.

Ни в коем случае нельзя использовать радиомодуль для региона 1 (433 МГц) в странах региона 2 (например, в Соединенных Штатах), поскольку в этих странах частота 433 МГц зарезервирована для работы лицензированных любительских радиостанций. Даже если поставщик предоставляет радиомодуль для одного региона в другом регионе, ответственность за понимание и следование местным правилам лежит на пользователе. Поэтому обязательно ознакомьтесь с документацией на используемое устройство и местными законами. Некоторая базовая информация по этому вопросу приводится в этой статье на Wikipedia: **https://oreil.ly/Adh7k**.

В функции `setup()` осуществляется несколько инициализаций, включая настройку контактов для встроенного светодиода и линии сброса радиомодуля. Затем выполняется сброс настроек радиомодуля: подачей на линию сброса низкого уровня, затем — высокого уровня, а затем — снова низкого. После этого производится настройка параметров радиомодуля.

В главном цикле `loop()` скетч посылает сообщение другому узлу, используя метод `sendtoWait()`. Затем скетч ожидает ответа от узла приемника, используя для этого функцию `recvfromAckTimeout()`. Получив ответ от узла приемника, скетч отображает его в окне монитора порта и мигает светодиодом.

Скетч для узла приемника очень похож на скетч для узла передатчика, за исключением главного цикла `loop()`, который ожидает получение сообщения, периодически проверяя значение метода `available()`. Получив сообщение, он отображает его в окне монитора порта, мигает светодиодом и посылает ответ узлу передатчика.

В процессе исполнения обоих скетчей в окне монитора порта, подключенного к узлу передатчика, должно отображаться следующее содержимое:

```
Resetting radio                  // Сбрасываем настройки радиомодуля
RFM69 radio running at 915 MHz   // Модуль RFM69 работает
                                 // на частоте 915 МГц
Got [Goodbye!] from 1            // Получили [Goodbye!] от 1
Got [Goodbye!] from 1
Got [Goodbye!] from 1
```

А в окне монитора порта, подключенного к узлу приемника, должен выводиться следующий текст:

```
Resetting radio
RFM69 radio running at 915 MHz
Got [Hello!] from 2
Got [Hello!] from 2
Got [Hello!] from 2
```

При исполнении любого из этих двух скетчей на 32-разрядной плате, плате Leonardo или на плате, которая не выполняет сброс при открытии последовательного порта, можно добавить строку кода:

```
while(!Serial);
```

сразу же после строки:

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

При этом исполнение скетча приостановится (см. *разд. «Особенности поведения аппаратных последовательных портов» главы 4*), пока не будет подключен монитор порта. Но по этой же причине такая строка кода весьма нежелательна в устройствах для промышленного применения, поскольку в случае отсутствия последовательного подключения скетч никогда не продолжит исполнение.

Сборка нескольких байтов сообщения в пакеты выполняется средствами библиотеки RadioHead, поэтому задача отправки двоичных данных состоит лишь в передаче адреса данных и количества отправляемых байтов.

Скетч, приведенный далее в листинге 14.4, очень похож на скетч передатчика из листинга 14.1 с той лишь разницей, что вместо передачи строки, он заполняет буфер сообщения содержимым структуры (`struct`) `sensor`, состоящим из значений аналоговых сигналов датчиков, считанных с трех входных аналоговых контактов. Первый элемент структуры представляет собой заголовок, посредством которого можно указывать тип сообщения. На случай использования передатчиком и приемником разных архитектур в структуре для представления аналоговых значений применен тип данных `short int`. Значение типа `int` занимает два байта на 8-разрядных платах и четыре байта на 32-разрядных, но безнаковые значения типа `short int` занимают два байта в обеих этих архитектурах. Однако выравнивание структур осуществляется по-разному в платах с разными архитектурами. Эта проблема решается использованием элемента дополнения структуры `padding`, который предназначен исключительно для этой цели. При отсутствии этого элемента размер структуры на 8-разрядных платах будет семь байтов, а на 32-разрядных — восемь. В результате этого данные на конце узла приемника будут искажаться (см. *разд. 4.6*).

Передаваемые по радио структуры определяются вне функции `loop()`, в результате чего они не попадают в стек этой функции. Такой подход требуется библиотеке RadioHead с тем, чтобы иметь возможность надежного доступа к памяти, в которой хранятся структуры.

Листинг 14.4. Передача данных в структуре

```
struct sensor
{
   char header = 'H';
   char padding; // Обеспечиваем одинаковое выравнивание
                 // для 8-битных и 32-битных платформ
   unsigned short int pin0;
   unsigned short int pin1;
   unsigned short int pin2;
} sensorStruct;

void loop()
{
   delay(1000); // Пауза длительностью в 1 секунду
```

```
   sensorStruct.pin0 = analogRead(A0);
   sensorStruct.pin1 = analogRead(A1);
   sensorStruct.pin2 = analogRead(A2);

   byte len = sizeof(sensorStruct);
   memcpy(message, &sensorStruct, len);

   if (!rf69_manager.sendtoWait((byte *)message, len, DEST_ADDR))
   {
      Serial.print("Failed to send message to "); Serial.println(DEST_ADDR);
                  Не удалось отправить сообщение
   }
}
```

А в листинге 14.5 приводится соответственно модифицированный скетч приемника.

Листинг 14.5. Прием данных в структуре

```
struct sensor
{
char header;
char padding; // Обеспечиваем одинаковое выравнивание
              // для 8-битных и 32-битных платформ
unsigned short int pin0;
unsigned short int pin1;
unsigned short int pin2;
} sensorStruct;

// Определяем любые данные, передаваемые по радиоканалу,
// вне функции loop(), чтобы они не хранились в стеке
byte message[sizeof(sensorStruct)];

void loop()
{
   if (rf69_manager.available()) // Получили сообщение
   {
      byte len = sizeof(message);
      byte sender; // Номер ID отправителя
      // Ожидаем сообщение
      if (rf69_manager.recvfromAck(message, &len, &sender))
      {
         memcpy(&sensorStruct, message, len);
         Serial.print("Header: "); Serial.println(sensorStruct.header);
         Serial.print("Sensor 0: "); Serial.println(sensorStruct.pin0);
         Serial.print("Sensor 1: "); Serial.println(sensorStruct.pin1);
         Serial.print("Sensor 2: "); Serial.println(sensorStruct.pin2);
      }
   }
}
```

В мониторе порта узла приемника полученные значения аналоговых сигналов отобразятся следующим образом:

```
Header: H
Sensor 0: 289
Sensor 1: 288
Sensor 2: 281
Header: H
Sensor 0: 287
Sensor 1: 286
Sensor 2: 280
```

Следует иметь в виду, что максимальный размер буфера для объекта `RH_RF69` составляет 60 байтов (константа `RH_RF69_MAX_MESSAGE_LEN` определена в заголовочном файле библиотеки).

Радиус действия радиомодулей RFM69HCW не превышает 500 метров — в зависимости от величины напряжения питания и антенны, и уменьшается при наличии препятствий между приемником и передатчиком.

Дополнительная информация

Библиотека RadioHead также поддерживает беспроводную сетевую технологию с бóльшим радиусом действия и низким энергопотреблением LoRa. При оптимальных рабочих условиях модуль LoRa с характеристиками, подобными характеристикам рассмотренных в этом разделе радиомодулей RFM69HCW, будет иметь несколько больший радиус действия, чем эти устройства. Радиомодули LoRa RFM95W предлагаются компанией Adafruit (артикул 3072) и компанией SparkFun (артикул WRL-14916).

Справочные листки для модулей передатчика и приемника RFM69HCW можно загрузить здесь: **https://oreil.ly/xdXXY** или **https://oreil.ly/888zi**.

14.2. Подключение плат Arduino по сети ZigBee (802.15.4)

ЗАДАЧА

Требуется организовать взаимодействие плат Arduino по беспроводной сети ZigBee (сети стандарта 802.15.4).

Стандарт 802.15.4 Института IEEE (Institute of Electrical and Electronics Engineers — Институт инженеров по электротехнике и радиоэлектронике) охватывает маломощные цифровые радиоустройства — такие как, например, недорогие радиомодули XBee компании Digi International. Альянс компаний ZigBee поддерживает и публикует одноименный стандарт ZigBee для беспроводных персональных сетей. Стандарт ZigBee основан на стандарте IEEE и является его расширенной версией. Стандарт связи ZigBee реализуется многими радиоустройствами, включая радиомодули XBee компании Digi International.

Только те радиомодули XBee, для которых указано, что они являются совместимыми со стандартом ZigBee (например, модули XBee 3), гарантированно соответствуют стандарту ZigBee. При этом вы можете использовать подмножество функций (IEEE 802.15.4) стандарта ZigBee даже со старыми модулями XBee Series 1. Более того, большинство рассматриваемых в этой главе решений будут работать с радиомодулями XBee серии 1.

Устранение проблем при работе с радиомодулями XBee

В случае возникновения проблем с взаимодействием радиомодулей XBee убедитесь, что они оба принадлежат к одному и тому же семейству продуктов (Product family) и имеют одинаковый набор функциональностей (Function set) — например, семейство продуктов XB3-24 и набор функциональностей Digi XBee 3 Zigbee 3.0 TH, как показано далее на рис. 14.4, а также, что они оба имеют самую последнюю версию микропрограммного обеспечения (Firmware version). Если это не решит проблему, попробуйте использовать чуть более раннюю версию микропрограммного обеспечения для обоих радиомодулей.

На странице **Common XBee Mistakes** (Распространенные ошибки при работе с устройствами XBee) веб-сайта Роберта Фалуди (Robet Faludi) приводится обширный набор советов по поиску и устранению неполадок радиомодулей Xbee (см. **https://oreil.ly/YSoWL**). А в его книге «Building Wireless Sensor Networks» («Построение беспроводных сенсорных сетей») (издательство O'Reilly) приведена подробная информация по работе с этими радиомодулями.

РЕШЕНИЕ

Приобретите два или более радиомодуля XBee одинакового типа (например, два модуля XBee 3, или два Series 2, или два Series 1, и т. п.) и настройте их, как показано далее в *разд. «Обсуждение работы решения и возможных проблем»*, для взаимодействия друг с другом. Это нужно сделать перед тем, как подключать их физически к плате Arduino. Подключите один из них к плате Arduino, а другой — к компьютеру (как это сделать, также рассказано в *разд. «Обсуждение работы решения и возможных проблем»*). Подключение радиомодуля XBee (установленного на адаптерной плате) к плате Arduino показано на рис. 14.2. Обратите внимание на то, что контакт RX платы Arduino подключен к контакту TX радиомодуля XBee, и наоборот.

В листинге 14.6 приводится простой скетч для работы с двумя радиомодулями XBee. Скетч просто принимает сообщения, передаваемые ему другим радиомодулем XBee, и отправляет их обратно этому модулю. В скетче используйте определение `MYSERIAL`, соответствующее вашей плате (подробно об этом рассказано в *разд. «Аппаратные средства для последовательной связи» главы 4*).

Листинг 14.6. Скетч для простой связи между двумя радиомодулями XBee

```
/*
* Скетч XBee Echo
* Просто отправляет полученные сообщения обратно
*/

// Раскомментирована должна быть только одна из следующих двух строк кода
#define MYSERIAL Serial // Для плат Uno, Nano и других плат с микроконтроллером AVR
```

```
//#define MYSERIAL Serial1 // Для плат Nano Every, Uno WiFi R2,
                           // Leonardo и плат с микроконтроллером ARM

void setup()
{
   MYSERIAL.begin(9600);
   pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
   while (MYSERIAL.available() )
   {
      MYSERIAL.write(MYSERIAL.read()); // Посылаем полученное сообщение обратно
      digitalWrite(LED_BUILTIN, HIGH); // Мигаем светодиодом для индикации активности
      delay(10);
      digitalWrite(LED_BUILTIN, LOW);
      delay(10);
   }
}
```

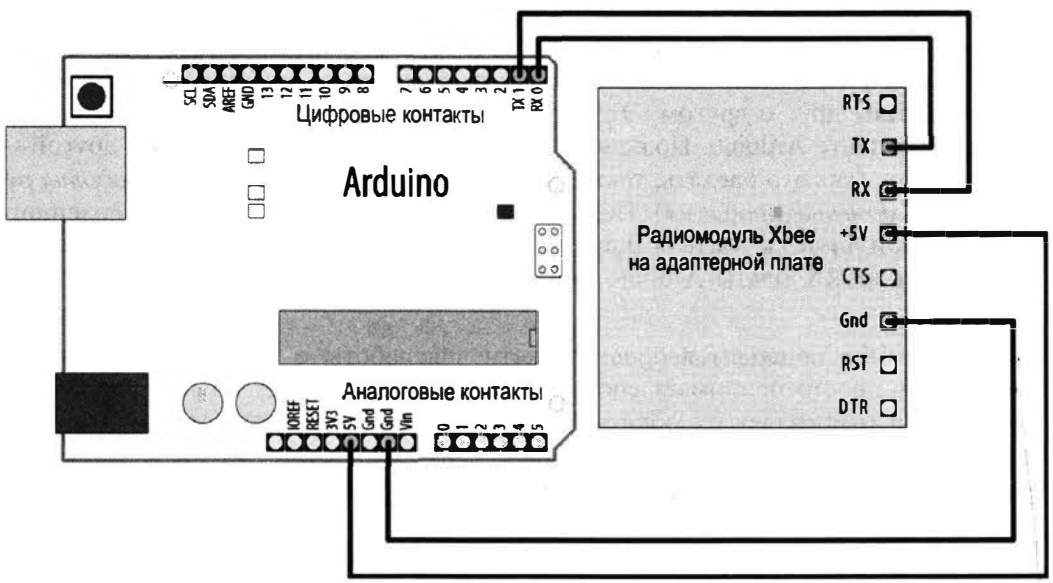


Рис. 14.2. Подключение радиомодуля Xbee, установленного на адаптерной плате, к плате Arduino

При соединении радиомодуля XBee с адаптерной платой, которая не оснащена стабилизатором напряжения, напряжение питания с платы будет подаваться напрямую на радиомодуль. В таком случае следует использовать стабилизатор напряжения, преобразующий напряжение величиной 5 В в напряжение величиной 3,3 В, поскольку контакт 3V3 платы Arduino не может поставлять ток достаточной

мощности для работы радиомодуля XBee (более подробно об этом рассказано далее в *разд. «Обсуждение работы решения и возможных проблем»*.

Выполнив настройку радиомодулей и подключив один из них к плате Arduino, а другой — к компьютеру, можно приступать к обмену сообщениями между ними.

Если вы работаете с платой Arduino Uno или любой другой платой, в которой цифровые контакты 0 и 1 (RX и TX) используются также и для интерфейса USB/RS-232, то, прежде чем загружать скетч в плату Arduino, радиомодуль нужно от нее отключить. Если этого не сделать, сигналы с компьютера будут смешиваться с сигналами радиомодуля.

Обсуждение работы решения и возможных проблем

Чтобы выполнить настройку радиомодуля XBee, его нужно вставить в адаптерную плату — например, в плату USB XBee Adapter компании Parallax (артикул 32400) или в плату XBee Explorer USB компании SparkFun (артикул WRL-11812) — и подключить эту адаптерную плату к разъему USB компьютера под управлением Windows, Linux или macOS. Кроме подключения к компьютеру с помощью такой адаптерной платы радиомодуль XBee можно подключать и к беспаечной макетной плате (штыревые разъемы этих радиомодулей не подходят к гнездовым разъемам беспаечных макетных плат). Если вы уже подключили радиомодули XBee к плате Arduino, то, прежде чем подключать их к компьютеру через USB-порт для выполнения настроек, их нужно отключить от плат Arduino (подключение радиомодулей к компьютеру требуется только для выполнения первичной их конфигурации).

На рынке предлагаются адаптерные платы для радиомодулей XBee без разъема USB. Такие адаптерные платы позволяют подключать радиомодуль к макетной плате, но для подключения к компьютеру модуля, установленного на такой макетной плате, необходимо использовать переходник USB/RS-232. Вы можете, как вариант, приобрести только одну адаптерную плату с USB-разъемом для конфигурирования радиомодулей и требуемое количество более дешевых адаптерных плат без такого разъема для подключения модулей к плате Arduino или другим устройствам. Но с этим подходом может возникнуть проблема, состоящая в том, что более дешевые адаптерные платы часто не содержат стабилизатора напряжения, выдающего напряжение величиной 3,3 В, необходимого для питания радиомодуля. А подключать контакт питания адаптерной платы к контакту 5V платы Arduino нельзя. Также не получится и подключить его к контакту 3V3 платы Arduino, поскольку этот контакт не может предоставить достаточно тока для питания радиомодуля XBee. Поэтому для питания радиомодуля потребуется использовать отдельный стабилизатор напряжения (например, LD1117V33). С таким стабилизатором напряжения также нужно будет использовать два блокирующих конденсатора: один (емкостью 10 мкФ) подключается между линией входного напряжения величиной 5 В и «землей», а другой (емкостью 1 мкФ) — между линией выходного напряжения 3,3 В и «землей» (подробная информация по организации питания радиомодуля XBee таким образом приводится в *разд. 15.5*, а на рис. 15.1 показана схема подключения стабилизатора напряжения LD1117V33 и блокирующих конденсаторов).

В случае подключения радиомодуля XBee к плате Arduino с напряжением питания 5 В может оказаться полезным подключать сигнальные линии TX и RX радиомодуля к соответствующим контактам платы Arduino через преобразователь логических уровней, хотя это и не обязательно. Согласно мнению Роберта Фалуди, бывшего главного разработчика компании Digi International (которая производит радиомодули XBee), радиомодули XBee «могут работать с логическими уровнями величиной 5 В, поступающими с платы Arduino». Он также считает, что если даже схема преобразования логических уровней и не всегда необходима, ее рекомендуется использовать в «промышленных приложениях, для которых требуется стабильная работа модуля во всем его рабочем температурном диапазоне» (см. **https://oreil.ly/eqzQ7**).

Рекомендуется приобрести как минимум две адаптерные платы с разъемом USB, чтобы можно было одновременно подключать к компьютеру два радиомодуля XBee. Эти же адаптерные платы можно также использовать и для подключения радиомодулей XBee к плате Arduino.

Конфигурирование радиомодуля XBee

Для первоначальной настройки модулей XBee их нужно подключить к компьютеру. Подключите один из модулей сейчас. Для правильной работы адаптерной платы с USB-разъемом может потребоваться установить драйверы для нее, поэтому посетите веб-страницу производителя вашей адаптерной платы и проверьте, требуются ли для нее драйверы, и если требуются, то загрузите их совместно с инструкцией по их установке. В большинстве таких адаптерных плат используется чипсет FTDI (об установке драйверов FTDI рассказано в *разд. 1.1*).

Прежде чем продолжать, загрузите и установите приложение XCTU для конфигурирования радиомодулей XBee (**https://oreil.ly/UTYaR**). Приложение поддерживается всеми тремя основными платформами: Windows, Linux и macOS.

Затем выполните начальные настройки для каждого радиомодуля XBee согласно следующей процедуре:

1. Запустите на исполнение приложение XCTU и выберите в меню **XCTU** опцию **Discover Radio Modules** (Откройте радиомодули) — откроется список доступных последовательных портов компьютера. Выберите порт, к которому, как вы полагаете, подключен радиомодуль XBee (возможно, лучше всего выбрать их все с помощью опции **Select All**). Нажмите кнопку **Next**.
2. Вам будет предложено задать параметры порта, определяющие, каким образом приложение XCTU будет пытаться взаимодействовать с подключенными радиомодулями. Если вы не изменили настройки по умолчанию своего радиомодуля на другие значения скорости обмена в бодах, количества битов данных, использования бита четности, использования и количества стоповых битов или управления потоком данных, для параметров порта можно оставить их значения по умолчанию. В противном случае установите значения, соответствующие значениям этих же параметров радиомодуля XBee. Нажмите кнопку **Finish** — приложение XCTU запустит процесс сканирования для обнаружения подключенных радиомодулей XBee.

3. По завершении сканирования приложение XCTU отобразит список обнаруженных им радиомодулей XBee, и все они будут выбраны (отмечены галочками) по умолчанию (рис. 14.3). Оставьте галочку только у того радиомодуля, для которого нужно выполнить настройку, и уберите их со всех остальных обнаруженных радиомодулей, а затем нажмите кнопку **Add Selected Devices** (Добавить выбранные устройства).

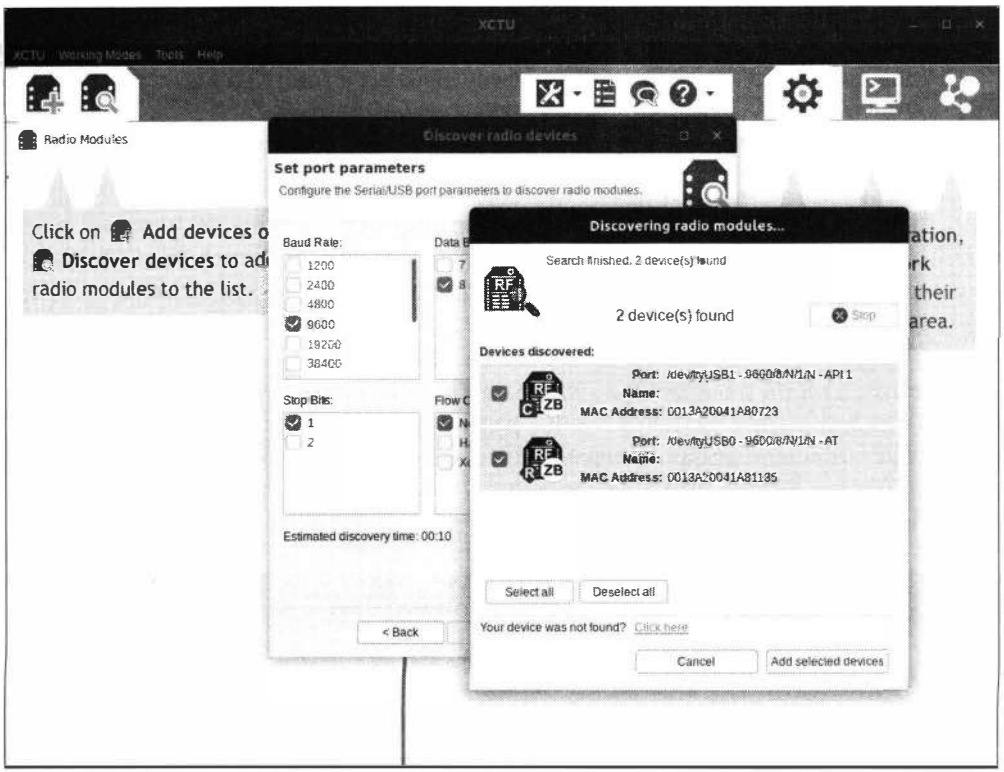


Рис. 14.3. Обнаружение программой XCTU радиомодулей XBee, подключенных к компьютеру

4. При первом подключении более ранних радиомодулей XBee (серии 1 или 2) приложение XCTU предложит заменить библиотеку микропрограммного обеспечения. В таком случае нажмите кнопку **Yes**, а затем установите пакет соответствующего модулю микропрограммного обеспечения. Затем приложение XCTU прочитает текущие настройки радиомодуля. Чтобы получить наилучшие результаты, для каждого из подключенных радиомодулей XBee сейчас (и каждый следующий раз при обновлении их конфигурационных настроек) нужно выполнить следующую процедуру:

   - выберите устройство, щелкнув на нем, а затем нажмите кнопку **Update** (Обновить) — откроется окно программы обновления микропрограммного обеспечения. Выберите в нем самую последнюю версию доступного микропрограммного обеспечения. Для радиомодулей XBee 3 — это опция **Digi XBee 3 Zigbee 3.0 TH**. Для каждого из радиомодулей XBee Series 2 потребуется раз-

ная версия микропрограммного обеспечения: для первого модуля — выберите опцию **ZigBee Coordinator AT**, а для второго — опцию **ZigBee Router AT**. Для радиомодулей XBee Series 1 выберите опцию **XBEE 802.15.4**. Нажмите кнопку **Update** (Обновить). Этот шаг можно пропустить, если уже используется самая последняя версия микропрограммного обеспечения.

- Нажмите кнопку **Default** (По умолчанию), а затем кнопку **Write** (Записать) — для устройства будут установлены настройки по умолчанию.

В случае возникновения каких-либо проблем с установкой и использованием приложения XCTU инструкции по их разрешению вы найдете на веб-странице: **https://oreil.ly/wrMpb**.

Пометьте первый сконфигурированный модуль каким-либо образом, чтобы его можно было отличить от другого модуля. Например, наклейте на него отрезок клейкой ленты или этикетку и напишете на ней 1, или A, или любое другое обозначение, указывающее, что это первый сконфигурированный радиомодуль. Теперь подключите к компьютеру второй радиомодуль XBee и повторите ту же самую процедуру (при этом первый модуль можно оставить подключенным к компьютеру). Пометьте второй радиомодуль каким-либо образом, чтобы его можно было отличить от первого (при использовании радиомодулей XBee Series 2 на первый из них будет установлено микропрограммное обеспечение ZigBee Coordinator AT, а на второй — ZigBee Router AT). Теперь все готово для выполнения последнего этапа конфигурирования (рис. 14.4).

Рис. 14.4. Индивидуальное конфигурирование радиомодулей XBee

1. Щелкните на первом радиомодуле XBee в окне программы XCTU.

   Для радиомодулей версии XBee 3 установите следующие настройки:

   - **CE** Device Role: Form Network [1] (Роль устройства CE: Создание сети [1]);
   - **ID** Extended Pan ID: 1234 (или любое другое шестнадцатеричное число, при условии использования одинакового идентификатора PAN ID для всех устройств одной радиосети);
   - **DH** Destination Address High: 0;
   - **DL** Destination Address Low: FFFF (это позволит радиомодулю-координатору передавать радиомодулю-маршрутизатору).

   Для радиомодулей XBee Series 2 установите следующие настройки:

   - **ID** Pan ID: 1234;
   - **DH** Destination Address High: 0;
   - **DL** Destination Address Low: FFFF.

   Для радиомодулей XBee Series 1 установите следующие настройки:

   - **ID** Pan ID: 1234;
   - **DH** Destination Address High: 0;
   - **DL** Destination Address Low: 2222;
   - **MY** 16-Bit Source Address: 1111.

   Команда **MY** устанавливает идентификатор для радиомодуля XBee. Команды **DL** и **DH** устанавливают младший и старший байты адреса радиомодуля назначения. Команда **ID** устанавливает ID-идентификатор радиосети (этот идентификатор должен быть одинаковым для всех устройств XBee Series 1, чтобы они могли взаимодействовать друг с другом).

2. Нажмите кнопку **Write**.
3. Щелкните на втором радиомодуле XBee в окне программы XCTU.

   Для радиомодулей версии XBee 3 установите следующие настройки:

   - **CE** Device Role: Join Network [0] (Роль устройства: Подключение к сети [0]);
   - **ID** Extended Pan ID: 1234;
   - **JV** Coordinator Verification: Enabled (таким образом обеспечивается подтверждение радиомодулем его нахождение на правильном канале, что делает более надежным его подключение к координатору).

   Для радиомодулей версии XBee 2 установите следующие настройки:

   - **ID** Pan ID: 1234;
   - **JV** Channel Verification: Enabled.

Для радиомодулей XBee Series 1 установите следующие настройки:

- **ID** Pan ID: 1234;
- **DH** Destination Address High: 0;
- **DL** Destination Address Low: 1111;
- **MY** 16-Bit Source Address: 2222.

4. Нажмите кнопку **Write**.

При наличии доступа к двум компьютерам можно подключить каждый радиомодуль XBee к отдельному компьютеру.

Далее щелкните на вкладке **Consoles** (со значком монитора) в правом верхнем углу окно XCTU или выполните команду меню **Working Modes | Consoles Working Mode** (Рабочие режимы | Рабочий режим консолей). Щелкните на каждом радиомодуле XBee в списке в левой панели, а затем щелкните на значке **Open** в правой панели вкладки **Consoles**. После подключения обоих радиомодулей XBee можно переключаться между ними и вводить текст непосредственно в журнал **Consol Log**. Текст, вводимый в консоль для одного радиомодуля, будет отображаться синим шрифтом. Щелчок мышью на другом радиомодуле XBee откроет его консоль, в которой текст, введенный в консоли первого радиомодуля, будет отображаться красным шрифтом.

Взаимодействие с платой Arduino

Выберите один из сконфигурированных радиомодулей XBee, закройте консоль, к которой он подключен, и отсоедините модуль от компьютера. Загрузите в плату Arduino скетч из листинга 14.6 и подключите к ней радиомодуль XBee, как показано на рис. 14.2. Введите какой-либо текст в консоль программы XCTU, к которой подключен другой радиомодуль XBee. Этот текст должен возвращаться в консоль первым радиомодулем. Иными словами, если ввести в консоль, например, букву «а», то в консоли отобразится текст «аа». Встроенный светодиод платы Arduino должен мигать при получении каждого символа от другого радиомодуля XBee.

Дополнительная информация

Дополнительный материал по работе с радиомодулями XBee приводится в *разд. 14.3–14.5*.

14.3. Обмен сообщениями с конкретным радиомодулем XBee

ЗАДАЧА

Требуется отправлять сообщение скетчем Arduino определенному радиомодулю XBee.

РЕШЕНИЕ

Конфигурационные команды с приставкой AT (attention, внимание) можно отправлять непосредственно со скетча Arduino. Это стандартный способ передачи команд управления устройствам, подключенным через последовательный порт. В листинге 14.7 приводится код скетча, который отправляет такие команды.

Листинг 14.7. Передача скетчем Arduino команд определенному радиомодулю XBee

```
/*
* Скетч XBee Message
* Отправляет сообщение радиомодулю XBE, используя его адрес
*/

// Раскомментирована должна быть только одна из следующих двух строк кода
#define MYSERIAL Serial // Для плат Uno, Nano и других плат
                        // с микроконтроллером AVR
//#define MYSERIAL Serial1 // Для плат Nano Every, Uno WiFi R2,
                           // Leonardo и плат с микроконтроллером ARM

bool configured;

bool configureRadio()
{
   // Устанавливаем для радиомодуля командный режим:
   MYSERIAL.flush();
   MYSERIAL.print("+++");
   delay(100);

   String ok_response = "OK\r"; // Ожидаемый ответ

   // Считываем текст ответа в переменную response
   String response = String("");
   while (response.length() < ok_response.length())
   {
      if (MYSERIAL.available() > 0)
      {
         response += (char) MYSERIAL.read();
      }
   }

   // Если получили правильный ответ, конфигурируем радиомодуль
   // и возвращаем значение true
   if (response.equals(ok_response))
   {
      MYSERIAL.print("ATDH0013A200\r"); // Старшая часть адреса модуля
            // назначения – ЗАМЕНИТЕ 0013A200 соответствующим адресом
```

```
      delay(100);
      MYSERIAL.print("ATDL403B9E1E\r"); // Младшая часть адреса модуля
            // назначения – ЗАМЕНИТЕ 403B9E1E соответствующим адресом
      delay(100);
      MYSERIAL.print("ATCN\r"); // Переводим радиомодуль обратно
                               // в режим данных
      return true;
   }
   else
   {
      return false; // Означает, что получен неправильный ответ
   }
}

void setup ()
{
   MYSERIAL.begin(9600); // Инициализируем последовательный интерфейс
   delay(1000);
   configured = configureRadio();
}

void loop ()
{
   if (configured)
   {
      MYSERIAL.print("Hello!");
      delay(3000);
   }
   else
   {
      delay(30000); // Пауза длительностью в 30 секунд
      configured = configureRadio(); // Повторяем попытку
   }
}
```

Обсуждение работы решения и возможных проблем

Хотя процесс конфигурирования, рассмотренный в *разд. 14.2*, хорошо работает с двумя радиомодулями XBee, он не дает достаточной гибкости для работы с бóльшим количеством устройств.

Рассмотрим, например, трехузловую сеть радиомодулей XBee Series 2 или XBee 3, в которой один радиомодуль сконфигурирован как координатор — Coordinator AT (или в случае устройств XBee 3 сконфигурирован для создания сети), а два других радиомодуля сконфигурированы как маршрутизаторы — Router AT (или для подключения к сети). Отправляемые координатором сообщения будут передаваться в широковещательном режиме двум маршрутизаторам, а сообщения, отправляемые каждым маршрутизатором, будут передаваться координатору.

Предложенная в решении этого раздела версия конфигурации для радиомодулей XBee Series 1 является несколько более гибкой в том смысле, что в ней указываются явные узлы назначения: сконфигурировав эти устройства с помощью команд AT, а затем записав полученную конфигурацию, вы, по сути, жестко кодируете адреса устройств в микропрограммное обеспечение.

Таким образом, текущее решение позволяет скетчу Arduino посылать команды для настройки радиомодулей XBee в процессе работы. Ключевым элементом решения является функция `configureRadio()`. Эта функция передает управляющую последовательность `++++`, которая переводит радиомодуль XBee в командный режим, подобно тому, как это делается в конце процесса конфигурирования радиомодуля XBee Series 1 (см. *разд. 14.2*). Отправив эту управляющую последовательность, скетч Arduino ожидает от модуля ответ OK, получив который он посылает модулю следующие команды AT:

```
ATDH0013A200
ATDL403B9E1E
ATCN
```

Первые две команды похожи на команды конфигурирования для радиомодулей XBee Series 1, которые мы выполняли в конце процесса конфигурирования в *разд. 14.2*, но здесь значения длиннее. Это объясняется тем, что в этом решении используются адреса в стиле радиомодулей XB Series 2. Как отмечалось в *разд. 14.2*, адрес для радиомодулей XBee Series 1 можно задать посредством команды AT MY, но радиомодули XBee Series 2 оснащены однозначным адресом, прошитым в микросхеме модуля.

В текущем решении нам нужно заменить значения `0013A200` (DH) и `403B9E1E` (DL) требуемыми соответствующими верхними (High) и нижними (Low) частями адреса. Эти данные для конкретного радиомодуля XBee можно узнать по его серийному номеру с помощью приложения XCTU, как показано на рис. 14.5. В частности, значение **SH** (Serial Number High) используется в качестве значения DH, а значение **SL** (Serial Number Low) — в качестве значения DL. Значения старшей (SH) и младшей (SL) частей серийного номера также указываются на наклейке внизу радиомодуля. Для радиомодулей XBee Series 1 используйте значение 0 для верхней части адреса устройства назначения, а для нижней части — адрес **MY** этого устройства.

Предлагаемое решение будет наиболее эффективным при наличии в сети третьего радиомодуля XBee с такими же настройками, как и у второго радиомодуля, настройки которого описываются в *разд. 14.2*. В случае использования радиомодулей XBee Series 1 устройству необходимо присвоить однозначный номер MY ID — например: 3333. Оставьте этот третий радиомодуль XBee подключенным к компьютеру, переключитесь на него в приложении XCTU в режим **Consoles** и нажмите кнопку **Open**. При исполнении скетча Arduino в окне консоли должно отобразиться сообщение **Hello!**.

Команда ATCN переводит радиомодуль из командного режима обратно в режим данных. Эту операцию можно рассматривать как обратную операции, выполняемой командой `++++`.

Рис. 14.5. Поиск старшей и младшей частей серийного номера радиомодуля XBee в приложении XCTU

Дополнительная информация

Дополнительная информация по работе с беспроводной связью приводится в *разд. 14.2*.

14.4. Обмен данными между радиомодулями XBee

ЗАДАЧА

Требуется передать состояние цифровых и аналоговых контактов или контактов управления на основе команд, получаемых от радиомодуля XBee.

РЕШЕНИЕ

Подключите один из радиомодулей XBee (тот, который сконфигурирован на передачу) к аналоговому датчику и настройте его на периодическое считывание показаний датчика и передачу полученного значения. Подключите к плате Arduino принимающий радиомодуль XBee, сконфигурированный для работы в режиме API, и считывайте кадры API, которые он получает от передающего радиомодуля.

Обсуждение работы решения и возможных проблем

Радиомодули XBee оснащены встроенным аналого-цифровым преобразователем (АЦП), который можно опрашивать через регулярные интервалы времени. Соответственно радиомодуль XBee можно настроить для передачи выходных значений АЦП (в диапазоне от 0 до 1023) другим радиомодулям сети.

Конфигурирование радиомодулей

Используя приложение XCTU, настройте радиомодули XBee в порядке, описанном ранее в *разд. «Конфигурирование радиомодуля Xbee»*, но со следующими изменениями:

- Для радиомодулей XBee 3.

  Передающим радиомодулем является модуль, для которого задана роль устройства CE — Join Network (Присоединение к сети). Кроме настроек, заданных для передающего радиомодуля ранее в *разд. «Конфигурирование радиомодуля Xbee»* (CE=Join Network [0], ID=1234, JV=Enabled [1]), используя приложение XCTU, установите и запишите для него следующие параметры: в разделе **I/O Settings** задайте для параметра **D0** AD0/DIO0 Commissioning Button Configuration значение ADC [2], а для параметра **IR** Sampling Rate — значение 64 (шестнадцатеричное значение для 100 мс). Этот передающий радиомодуль XBee будет подключен к датчику.

- Для радиомодулей XBee Series 2:

  - установите в принимающий радиомодуль XBee микропрограммное обеспечение ZigBee Coordinator API вместо ZigBee Coordinator AT. Это позволит этому радиомодулю принимать кадры API. Значения остальных параметров остаются такими же, как и прежде (ID=1234, DL=FFFF). Этот принимающий радиомодуль XBee будет подключен к плате Arduino;

  - другой (передающий) радиомодуль XBee мы запрограммировали микропрограммным обеспечением ZigBee Router AT (программировать его микропрограммным обеспечением ZigBee Coordinator API нет надобности). Кроме настроек, заданных для передающего радиомодуля ранее в *разд. «Конфигурирование радиомодуля Xbee»* (ID=1234, JV=Enabled [1]), используя приложение XCTU, установите и запишите для него следующие параметры: в разделе **I/O Settings** задайте для параметра **D0** AD0/DIO0 Configuration значение ADC [2], а для параметра **IR** Sampling Rate — значение 64 (шестнадцатеричное значение для 100 мс). Этот передающий радиомодуль XBee будет подключен к датчику.

- Для радиомодулей XBee Series 1.

  Передающим радиомодулем является модуль, для которого настроен 16-разрядный адрес источника (Source Address) MY, равный 2222. Кроме настроек, заданных для передающего радиомодуля ранее в *разд. «Конфигурирование радиомодуля Xbee»* (ID: 1234, DH: 0, DL: 1111, MY: 2222), используя приложение XCTU,

установите и запишите для него следующие параметры: в разделе **I/O Settings** задайте для параметра **D0** DIO0 Configuration значение ADC [2], а для параметра **IR** Sampling Rate — значение 100 (шестнадцатеричное значение для 100 мс). Этот передающий радиомодуль XBee будет подключен к датчику.

Для радиомодулей XBee 3 и XBee Series 2 значения адресов назначения Destination Address (DH, DL) устанавливать не нужно, поскольку маршрутизатор взаимодействует с координатором по умолчанию. При другой конфигурации радиосети для передающего радиомодуля XBee можно задать следующие дополнительные параметры, чтобы указать ему радиомодуль, которому следует отправлять данные:

- Destination Address High (DH): старший адрес (SH) другого (передающего) радиомодуля XBee, обычно это будет 13A200.
- Destination Address Low (DL): младший адрес (SL) другого радиомодуля XBee.

Теперь подключите принимающий радиомодуль XBee к плате Arduino, как показано на рис. 14.2 в *разд. 14.2*. Также нужно подключить светодиод (через токоограничивающий резистор, как это сделано в *разд. 7.2*) к контакту 5 и шине «земли» платы Arduino. Если используемая плата не поддерживает ШИМ-сигнал на контакте 5, используйте другой контакт с этой возможностью и внесите соответствующую корректировку в скетч.

В зависимости от версии используемых радиомодулей XBee, подключать их к датчикам нужно по-разному. При этом радиомодули XBee 3 и XBee Series 2 подключаются одинаково, а вот подключение модуля XBee Series 1 производится иначе. Код скетча для разных модулей также будет различаться. Для радиомодулей XBee Series 2 режим API определяется версией исполняемого микропрограммного обеспечения, а для модулей XBee Series 1 и XBee 3 в режим API можно переключиться, выполнив команду ATAP.

Прежде чем устанавливать радиомодуль XBee в адаптерную плату, проверьте соответствие контактов платы и модуля, поскольку назначение контактов на некоторых адаптерных платах не всегда совпадает с контактами самого модуля XBee. Например, в некоторых адаптерных платах верхний левый контакт используется для подключения «земли», а контакт под ним — для подключения питания 3,3 В. Точно так же на некоторых адаптерных платах контакт VREF (обозначенный RES на адаптерной плате XBee Explorer USB компании SparkFun) пятый снизу на правой стороне платы, тогда как на самом модуле XBee он четвертый снизу.

Подключение к датчику передающих радиомодулей XBee 3 и XBee Series 2 показано на рис. 14.6. Значение номинала резистора R1 должно быть вдвое большим, чем значение номинала используемого потенциометра (датчика), — например, при номинале потенциометра 10 кОм номинал этого резистора должен быть 20 кОм. Такое требование объясняется тем, что диапазон напряжений входного сигнала аналого-цифрового преобразователя радиомодуля XBee Series 2 составляет от 0 до 1,2 В и резистор R1 понижает напряжение 3,3 В до уровня ниже 1,2 В.

Подключение к датчику передающего радиомодуля XBee Series 1 показано на рис. 14.7.

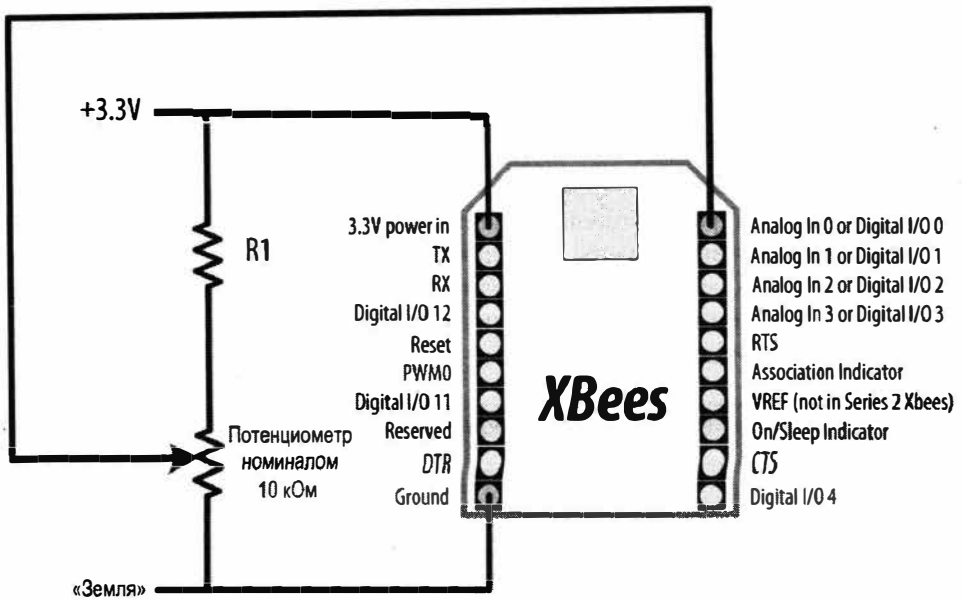


Рис. 14.6. Подключение к датчику передающих радиомодулей XBee 3 и XBee Series 2

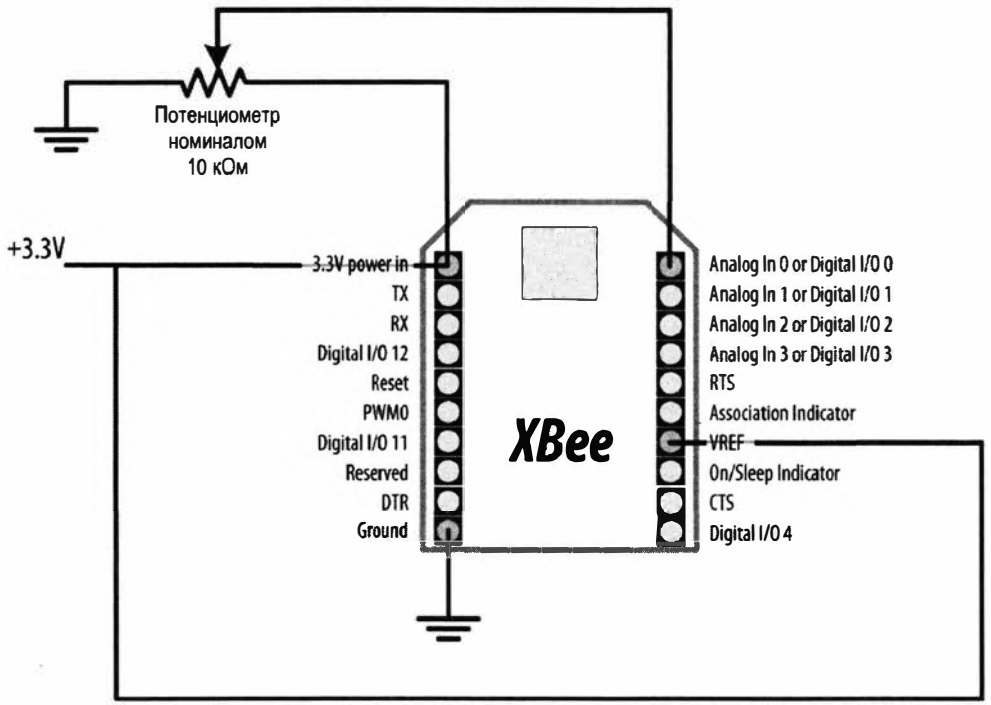


Рис. 14.7. Подключение к датчику передающего радиомодуля XBee Series 1

В отличие от радиомодулей XBee 3 и XBee Series 2, в радиомодуле XBee Series 1 используется внешний источник эталонного напряжения, подключаемый к напряжению питания 3,3 В. Поскольку напряжение на подвижном контакте потенциометра никогда не может быть больше, чем эталонное напряжение, резистор R1, используемый в схеме на рис. 14.6, здесь не требуется.

Для радиомодулей XBee Series 2 загрузите в плату скетч из листинга 14.8 и подключите передающий радиомодуль к плате Arduino, как показано на рис. 14.2 в *разд. 14.2*. Эти действия должны выполняться именно в такой последовательности, поскольку подключенный к плате радиомодуль будет мешать загрузке скетча. По этой же причине при необходимости повторной загрузки скетча не забудьте отключить от нее радиомодуль.

Листинг 14.8. Скетч для передающего радиомодуля XBee Series 2

```
/*
* Скетч XBeeAnalogReceive Series 2
* Считывает значения аналогового сигнала с кадра API Xbee
  и устанавливает соответствующий уровень яркости светодиода
*/

// Раскомментирована должна быть только одна из следующих двух строк кода
#define MYSERIAL Serial     // Для плат Uno, Nano и других плат с микроконтроллером AVR
//#define MYSERIAL Serial1 // Для плат Nano Every, Uno WiFi R2, Leonardo
                           // и плат с микроконтроллером ARM

#define MIN_CHUNK 24
#define OFFSET 18

const int ledPin = A5; // Номер аналогового контакта для подключения
                       // выходного сигнала датчика

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

void loop()
{
   if (MYSERIAL.available() >= MIN_CHUNK) // Ждем, пока не получим
                                          // достаточный объем данных
   {
      if (MYSERIAL.read() == 0x7E) // Ограничитель начала кадра
      {
         // Пропускаем ненужные байты в кадрах API
         for (int i = 0; i < OFFSET; i++)
         {
            MYSERIAL.read();
         }
```

```
        // Следующие два байта – старший и младший байты значения
        // считанного аналогового сигнала
        int analogHigh = MYSERIAL.read();
        int analogLow = MYSERIAL.read();
        int analogValue = analogLow + (analogHigh * 256);

        // Масштабируем значение яркости к диапазону
        // значений ШИМ платы Arduino
        int brightness = map(analogValue, 0, 1023, 0, 255);

        // Включаем светодиод
        analogWrite(ledPin, brightness);
    }
  }
}
```

Для радиомодулей XBee 3 и XBee Series 1 загрузите в плату скетч из листинга 14.9. Загрузку нужно выполнить перед подключением радиомодуля к плате, поскольку подключенный к плате радиомодуль будет мешать загрузке скетча. По этой же причине при необходимости повторной загрузки скетча не забудьте отключить от нее радиомодуль. Если используется радиомодуль XBee Series 1, закомментируйте значения `MIN_CHUNK` и `OFFSET` для радиомодуля XBee 3 и раскомментируйте значения для модуля XBee Series 1.

Листинг 14.9. Скетч для передающих радиомодулей XBee 3 и XBee Series 1

```
/*
* Скетч XBeeAnalogReceive Series 1 or XBee 3
* Считывает значения аналогового сигнала с кадра API Xbee
  и устанавливает соответствующий уровень яркости светодиода
*/

// Раскомментирована должна быть только одна из следующих двух строк кода
#define MYSERIAL Serial     // Для плат Uno, Nano и других плат
                            // с микроконтроллером AVR
//#define MYSERIAL Serial1  // Для плат Nano Every, Uno WiFi R2, Leonardo
                            // и плат с микроконтроллером ARM

// Для радиомодуля XBee 3 используйте эти параметры
#define MIN_CHUNK 21
#define OFFSET 18

// Для радиомодуля XBee Series 1 используйте эти параметры
//#define MIN_CHUNK 14
//#define OFFSET 10

const int ledPin = A5;
```

```
void setup()
{
   MYSERIAL.begin(9600);
   delay(1000);
   configureRadio(); // Если требуется обработка ошибок, выполняется
                     // проверка возвращаемого значения
}

bool configureRadio()
{
   // Устанавливаем для радиомодуля командный режим:
   MYSERIAL.flush();
   MYSERIAL.print("+++");
   delay(100);

   String ok_response = "OK\r"; // Ожидаемый ответ

   // Считываем текст ответа в переменную response
   String response = String("");
   while (response.length() < ok_response.length())
   {
      if (MYSERIAL.available() > 0)
      {
         response += (char) MYSERIAL.read();
      }
   }

   // Если получили правильный ответ, конфигурируем радиомодуль
   // и возвращаем значение true
   if (response.equals(ok_response))
   {
      MYSERIAL.print("ATAP1\r"); // Переходим в режим API
      delay(100);
      MYSERIAL.print("ATCN\r");  // Переводим радиомодуль обратно в режим данных
      return true;
   }
   else
   {
      return false; // Означает, что получен неправильный ответ
   }
}

void loop()
{
   if (MYSERIAL.available() >= MIN_CHUNK) // Ждем, пока не получим
                                          // достаточный объем данных
   {
      if (MYSERIAL.read() == 0x7E)        // Ограничитель начала кадра
```

```
    {
      // Пропускаем ненужные байты кадры API
      for (int i = 0; i < OFFSET; i++)
      {
        MYSERIAL.read();
      }

      // Следующие два байта – старший и младший байты значения
      // считанного аналогового сигнала
      int analogHigh = MYSERIAL.read();
      int analogLow = MYSERIAL.read();
      int analogValue = analogLow + (analogHigh * 256);

      // Масштабируем значение яркости к диапазону
      // значений ШИМ платы Arduino
      int brightness = map(analogValue, 0, 1023, 0, 255);

      // Включаем светодиод
      analogWrite(ledPin, brightness);
    }
  }
}
```

Код Arduino должен сконфигурировать радиомодули XBee Series 1 и XBee 3 для работы в режиме API, выполнив команду AT `ATAP1`. Для радиомодулей XBee Series 2 такое конфигурирование осуществляется прошивкой соответствующего микропрограммного обеспечения. Необходимость возврата в режим данных (`ATCN`) объясняется тем, что ранее был выполнен переход в командный режим, а для приема данных требуется, чтобы радиомодуль работал в режиме данных.

Дополнительная информация

Дополнительная информация по работе с беспроводной связью приводится также в *разд. 14.2*.

14.5. Активирование подключенного к радиомодулю XBee устройства

ЗАДАЧА

Требуется передать радиомодулю XBee команду активировать контакт, чтобы включить подключенное к нему устройство — например реле или светодиод.

РЕШЕНИЕ

Настройте подключенный к устройству радиомодуль XBee для работы в режиме приема, чтобы он мог получать инструкции от другого радиомодуля XBee. Под-

ключите другой (передающий) радиомодуль XBee к плате Arduino и загрузите в плату скетч для передачи команд, требуемых для активирования контакта принимающего радиомодуля, к которому подключено устройство. При этом не забудьте, что для загрузки скетча радиомодуль нужно отключить от платы Arduino, чтобы он не мешал загрузке.

Обсуждение работы решения и возможных проблем

Цифроаналоговые контакты ввода/вывода радиомодуля XBee можно настроить для подачи выходного цифрового сигнала. Кроме того, радиомодуль XBee можно настроить для приема инструкций от другого радиомодуля XBee, чтобы управлять этими контактами, устанавливая на них высокий или низкий уровень выходного сигнала. Для этого в радиомодулях XBee Series 2 используется функциональность Remote AT Command (дистанционные команды AT). А для радиомодулей XBee Series 1 можно организовать прямой ввод/вывод, создав виртуальную связь между радиомодулями.

Радиомодули XBee Series 2 и XBee 3

Используя приложение XCTU, сконфигурируйте принимающий радиомодуль XBee (см. ранее *разд.* «*Конфигурирование радиомодуля Xbee*»). К этому радиомодулю будет подключен светодиод, которым требуется управлять. Для радиомодуля XBee Series 2 выполните прошивку микропрограммного обеспечения ZigBee Router AT (не API). Далее выполните следующие настройки:

- (только для устройств XBee 3) **CE** Device Role: Join Network [0];
- **ID** Extended PAN ID: 1234 (или любое другое число при условии использования одинакового идентификатора PAN ID для обоих радиомодулей XBee);
- **JV** Channel Verification: Enabled [1];
- В разделе **I/O Settings**, **D1** DIO1/AD1/SPI_nATTN Configuration: Digital Out, Low [4].

Затем сконфигурируйте передающий радиомодуль XBee (который будет подключен к плате Arduino). Для радиомодуля XBee Series 2 выполните прошивку микропрограммного обеспечения ZigBee Coordinator API (не AT). Далее выполните следующие настройки:

- (только для устройств XBee 3) **CE** Device Role: Join Network [1];
- (только для устройств XBee 3) **AP** API Enable: API Mode Without Escapes [1];
- **ID** Extended PAN ID: 1234 (или любое другое число при условии использования одинакового идентификатора PAN ID для обоих радиомодулей XBee);
- **DH** Destination Address High: 0;
- **DL** Destination Address Low: FFFF.

Подключите принимающий модуль XBee к плате Arduino, как показано на рис. 14.8.

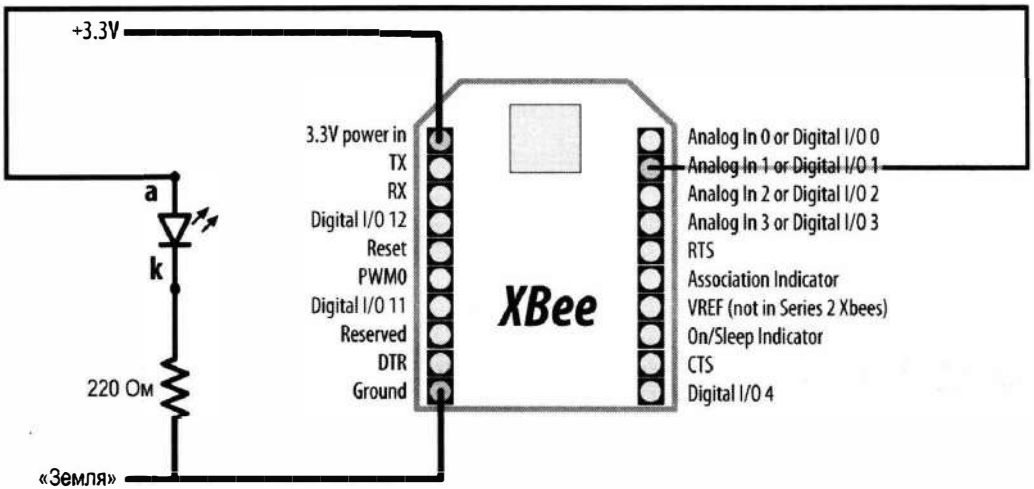


Рис. 14.8. Подключение светодиода к цифровому контакту радиомодулей XBee 3 и XBee Series 2

Затем загрузите в плату скетч из листинга 14.10 и подключите передающий радиомодуль к плате Arduino, как показано на рис. 14.2. Эти действия должны выполняться именно в такой последовательности, поскольку подключенный к плате радиомодуль будет мешать загрузке скетча. По этой же причине при необходимости повторной загрузки скетча не забудьте отключить от нее радиомодуль. Скетч отправляет дистанционную команду AT (`ATD14` или `ATD15`), которая поочередно то включает то выключает контакт 1 (`ATD1`), устанавливая на нем высокий (значение `0x5`) или низкий (значение `0x4`) логический уровень соответственно.

Листинг 14.10. Скетч для передачи команды управления выходным цифровым контактом радиомодулями XBee 3 и XBee Series 2

```
/*
Скетч XBeeActuate
Посылает дистанционную команду AT для активирования цифрового контакта
на другом радиомодуле XBee
*/

// Раскомментирована должна быть только одна из следующих двух строк кода
#define MYSERIAL Serial     // Для плат Uno, Nano и других плат
                            // с микроконтроллером AVR
//#define MYSERIAL Serial1  // Для плат Nano Every, Uno WiFi R2, Leonardo
                            // и плат с микроконтроллером ARM

const byte frameStartByte = 0x7E;
const byte frameTypeRemoteAT = 0x17;
const byte remoteATOptionApplyChanges = 0x02;

void setup()
```

```
{
   MYSERIAL.begin(9600);
}

void loop()
{
   toggleRemotePin(1);
   delay(2000);
   toggleRemotePin(0);
   delay(2000);
}

byte sendByte(byte value)
{
   MYSERIAL.write(value);
   return value;
}

void toggleRemotePin(int value) // 0 = выключен, не ноль = включен
{
   byte pin_state;
   if (value)
   {
      pin_state = 0x5;
   }
   else
   {
      pin_state = 0x4;
   }

   sendByte(frameStartByte); // Начало кадра API
   // Старшая и младшая части длины кадра (не считая контрольной суммы)
   sendByte(0x0);
   sendByte(0x10);

   long sum = 0; // Аккумулируем контрольную сумму

   sum += sendByte(frameTypeRemoteAT); // Указываем, что этот кадр
                              // содержит дистанционную команду AT

   sum += sendByte(0x0); // Задаем нулевой ID-номер кадра, означая,
                         // что ответ не требуется

   // Следующие 8 байтов указывают ID-номер получателя
   // Для широковещательной передачи на все узлы используйте 0xFFFF
   sum += sendByte(0x0);
   sum += sendByte(0x0);
```

```
    sum += sendByte(0x0);
    sum += sendByte(0x0);
    sum += sendByte(0x0);
    sum += sendByte(0x0);
    sum += sendByte(0xFF);
    sum += sendByte(0xFF);

    // Следующие 2 байта указывают 16-разрядный адрес получателя
    // Для широковещательной передачи на все узлы используйте 0xFFFE
    sum += sendByte(0xFF);
    sum += sendByte(0xFF);

    sum += sendByte(remoteATOptionApplyChanges); // Отправляет опции
                                                 // дистанционной команды AT

    // Текст команды AT
    sum += sendByte('D');
    sum += sendByte('1');

    // Значение (0x4 для выключить, 0x5 для включить)
    sum += sendByte(pin_state);

    // Отправляем контрольную сумму
    sendByte( 0xFF - ( sum & 0xFF));

    delay(10); // Пауза, чтобы дать микроконтроллеру возможность
               // стабилизироваться, если необходимо
}
```

Радиомодули XBee Series 1

Используя приложение XCTU, сконфигурируйте передающий радиомодуль XBee (который будет подключен к плате Arduino) следующим образом:

- **ID** Pan ID: 1234;
- **DH** Destination Address High: 0;
- **DL** Destination Address Low: 2222;
- **MY** 16-Bit Source Address: 1111;
- **D1** DIO1 Configuration: DI[3]. Этим задается цифровой входной режим работы для цифроаналогового ввода 1 радиомодуля XBee. Передающий радиомодуль XBee будет передавать состояние этого контакта принимающему радиомодулю;
- **IC** DIO Change Detect: FF. Этим дается указание радиомодулю XBee проверять состояние каждого цифрового контакта, работающего в режиме ввода, и передавать полученные значения радиомодулю XBee, обозначенному значениями `ATDL` и `ATDH`.

Затем выполните конфигурацию принимающего радиомодуля XBee:

- **ID** Pan ID: 1234;
- **DH** Destination Address High: 0;
- **DL** Destination Address Low: 1111;
- **MY** 16-Bit Source Address: 2222;
- **D1** DIO1 Configuration: DO Low [4]. Этим осуществляется настройка контакта 19 (цифроаналоговый ввод 1) для работы в выходном цифровом режиме (который по умолчанию выключен);
- **IU** I/O Output Enable: Disabled [0]. Этим радиомодулю XBee дается указание не посылать кадры, которые он получает по последовательному интерфейсу;
- **IA** I/O Input Address: 1111. Этим радиомодулю XBee дается указание принимать команды от другого радиомодуля (с адресом MY = 1111).

Подключите передающий радиомодуль XBee к плате Arduino, как показано на рис. 14.9.

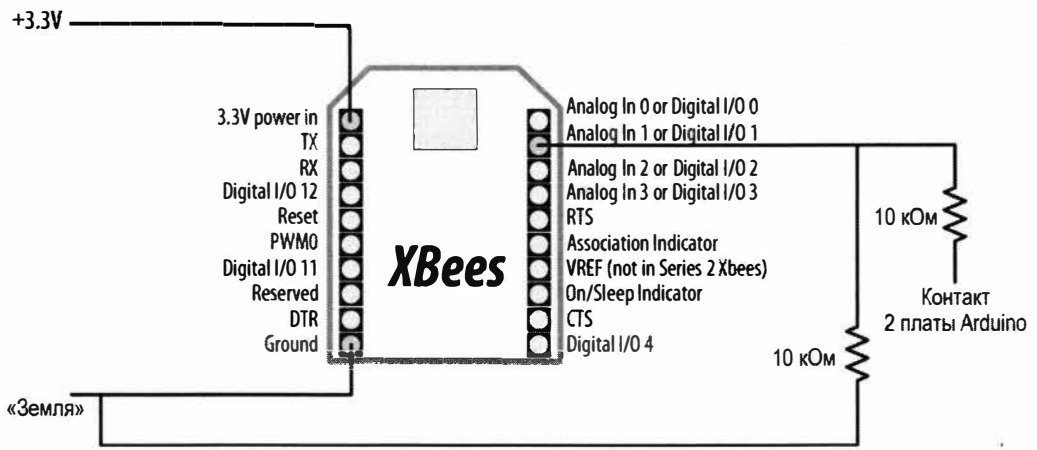


Рис. 14.9. Подключение радиомодуля XBee Series 1 к цифровому контакту 2 платы Arduino

Затем подключите светодиод к принимающему модулю XBee, как показано на рис. 14.8. Обратите внимание, что в нашем случае высокий уровень на контакте 1 радиомодуля XBee будет устанавливаться не командой AT, передаваемой через последовательный интерфейс, а с помощью электрического подключения. Показанные на рис. 14.9 два резистора номиналом 10 кОм создают делитель напряжения, который понижает уровень логического сигнала платы Arduino с 5 В до приблизительно 2,5 В. Это достаточно высокое напряжение, чтобы радиомодуль мог определять его как высокий логический уровень, но достаточно низкое, чтобы не повредить контакты радиомодуля XBee, рассчитанные на напряжение величиной 3,3 В.

Теперь загрузите скетч из листинга 14.11 в плату Arduino, к которой подключен передающий радиомодуль XBee. Скетч поочередно то включает, то выключает контакт 1 радиомодуля, устанавливая на нем высокий (значение `0x5`) или низкий

(значение 0x4) логический уровень соответственно. Поскольку передающий радиомодуль XBee настроен на передачу состояния своих контактов принимающему радиомодулю, то когда состояние его контакта 1 меняется, соответственно меняется и состояние контакта 1 принимающего модуля.

Листинг 14.11. Скетч для передачи команды управления выходным цифровым контактом радиомодулем XBee Series 1

```
/*
Скетч XBeeActuateSeries1
Управляет цифровым контактом на другом радиомодуле XBee Series 1
*/

const int xbeePin = 2;

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

void loop()
{
   digitalWrite(xbeePin, HIGH);
   delay(2000);
   digitalWrite(xbeePin, LOW);
   delay(2000);
}
```

Дополнительная информация

Дополнительная информация по работе с беспроводной связью приводится также в *разд. 14.2*.

14.6. Взаимодействие с классическими устройствами Bluetooth

ЗАДАЧА

Требуется реализовать обмен информацией платы Arduino с другим устройством — например, ноутбуком или смартфоном, используя Bluetooth.

РЕШЕНИЕ

Эту задачу можно решить, подключив к плате Arduino модуль Bluetooth — например, BlueSMiRF, Bluetooth Mate или Bluetooth Bee.

Скетч для управления модулем Bluetooth приводится в листинге 14.12. Этот скетч похож на скетч из решения *разд. 4.11* — он отслеживает входящие данные на аппаратном последовательном порте, либо на программном последовательном порте, либо на последовательном порте `Serial1`, подключенному к контактам TX/RX модуля Bluetooth. Таким образом, любые данные, получаемые по одному каналу, будут передаваться другому.

Подключение модуля к плате Arduino показано на рис. 14.10. При использовании платы Leonardo или других 32-разрядных плат раскомментируйте строку `#define USESERIAL1` и подключите радиомодуль Bluetooth к соответствующим контактам RX и TX платы (на платах форм-фактора платы Arduino Uno это контакты 0 и 1).

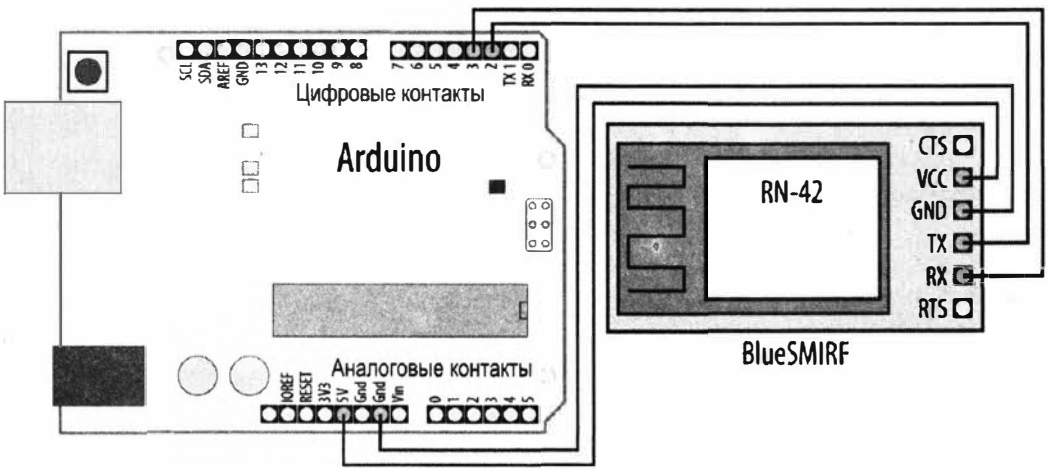


Рис. 14.10. Подключение радиомодуля Bluetooth BlueSMiRF к плате Arduino для взаимодействия через программный последовательный интерфейс

Листинг 14.12. Скетч для работы с модулем Bluetooth

```
/*
* Скетч Classic Bluetooth
* Взаимодействует с модулем Bluetooth по программному последовательному
  интерфейсу или по последовательному интерфейсу Serial1
* Код сопряжения обычно 1234
*/

//#define USESERIAL1 // Раскомментируйте эту строку, если используется
                     // плата Leonarldo или плата с микроконтроллером ARM
#ifdef USESERIAL1
   #define BTSERIAL Serial1
#else
   #include <SoftwareSerial.h>
   const int rxpin = 2;                  // Номер контакта платы для приема данных
   const int txpin = 3;                  // Номер контакта платы для передачи данных
```

```
   SoftwareSerial mySerial(rxpin, txpin); // Новый последовательный порт
                                          // на этих контактах
   #define BTSERIAL mySerial              // Программный последовательный интерфейс
#endif

void setup()
{
   Serial.begin(9600);
   BTSERIAL.begin(9600); // Инициализируем программный последовательный порт
   Serial.println("Serial ready");
   BTSERIAL.println("Bluetooth ready");
}

void loop()
{
   if (BTSERIAL.available())
   {
      char c = (char)BTSERIAL.read();
      Serial.write(c);
   }
   if (Serial.available())
   {
      char c = (char)Serial.read();
      BTSERIAL.write(c);
   }
}
```

Обсуждение работы решения и возможных проблем

Для взаимодействия с этим скетчем ваш компьютер должен поддерживать возможность связи по Bluetooth. Кроме того, для взаимодействия устройств по Bluetooth необходимо осуществить сопряжение их друг с другом — устройство Bluetooth, установленное на компьютере, должно знать ID-номер модуля Bluetooth, подключенного к плате Arduino. По умолчанию модулям Bluetooth типа BlueSMiRF присваивается пин-код 1234. Информацию по уточнению ID-номера сопряжения и разрешению соединения для устройства Bluetooth, установленного на компьютере, вам надо найти в документации на него.

После сопряжения компьютера с радиомодулем Bluetooth в списке его устройств станут отображаться несколько новых последовательных портов. На компьютере под Windows этот список находится в разделе **Порты** (COM & LPT) диспетчера устройств. Найдите в нем порт **Serial over Bluetooth Link** (Последовательный интерфейс по каналу Bluetooth). Если такой порт там не один, попробуйте их все. На компьютерах под macOS посмотреть список последовательных портов можно, выполнив в терминале следующую команду:

```
ls /dev/{cu,tty}.*
```

Требуемый порт будет иметь название типа: `/dev/cu.RN42-22AC-SPP`.

Все стандартные модули Bluetooth, используемые с платой Arduino, реализуют профиль Bluetooth SPP (Serial Port Profile, профиль последовательного порта). После выполнения сопряжения устройств компьютер будет определять такой модуль как последовательный порт. Эти модули не могут определяться как какой-либо иной тип устройства Bluetooth — например, как Bluetooth-мышь или Bluetooth-клавиатура.

Для подключения к последовательному порту Bluetooth компьютера можно использовать, например, программу терминала PuTTY для Windows (**https://oreil.ly/ZyOO**) или программу screen для Linux или macOS. Чтобы подключиться к последовательному порту на скорости 9600 бод, используя программу screen, откройте терминал и выполните в нем следующую команду (замените при этом идентификатор `tty.RN42-22AC-SPP` номером требуемого последовательного порта):

```
screen /dev/tty.RN42-22AC-SPP 9600
```

Радиус действия радиосвязи Bluetooth составляет от пяти до 10 метров — в зависимости от класса используемого радиомодуля: класс 3, 2 или 1.

Дополнительная информация

Компания SparkFun предоставляет учебное пособие по установке и использованию Bluetooth (**https://oreil.ly/jtwsm**).

Схема расположения контактов модуля Bluetooth Bee (**https://oreil.ly/ulf8K**) позволяет вставлять его в гнездовой разъем плат для радиомодулей XBee, что дает возможность использовать для модуля Bluetooth Bee шилды и адаптеры, выпускаемые для этих радиомодулей.

14.7. Работа с радиомодулями Bluetooth LE

ЗАДАЧА

Требуется реализовать обмен информацией платы Arduino с другим устройством, используя модуль Bluetooth LE (Low Energy — с низким энергопотреблением). Этот тип модулей представляет усовершенствованную, более гибкую и современную альтернативу классическим модулям Bluetooth. Работа технологии Bluetooth LE (BLE) во многом отличается от работы классической технологии Bluetooth и позволяет решать задачи другого типа. Тогда как технология Bluetooth Classic хорошо подходит для передачи сравнительного большого объема данных, технология BLE предназначена для использования с устройствами, посылающими короткие сообщения через более длительные интервалы времени, — например, с датчиками.

РЕШЕНИЕ

Для решения аппаратной части этой задачи можно использовать одну из многих плат Arduino, оснащенных встроенными возможностями BLE уровня Bluetooth 4.0 или выше. Например, плату Nano 33 BLE, Nano 33 IoT, Uno WiFi Rev 2 или MKR

WiFi 1010. Любая из этих плат поддерживает библиотеку ArduinoBLE (**https://oreil.ly/nWQ7O**), которая упрощает задачу использования технологии Bluetooth LE в проектах.

А программная часть задачи решается с помощью скетча из листинга 14.13. Это упрощенная версия одного из примеров скетча, входящих в состав библиотеки ArduinoBLE. Установите библиотеку с помощью менеджера библиотек и загрузите скетч в плату Arduino.

Листинг 14.13. Скетч для обмена информацией устройствами Bluetooth LE

```
/*
* Скетч ArduinoBLE
* Позволяет управление встроенным светодиодом с помощью
  модуля Bluetooth LE
*/

#include <ArduinoBLE.h>

#define SERVICE_ID "19B10010-E8F2-537E-4F6C-D104768A1214"
#define CHAR_ID "19B10011-E8F2-537E-4F6C-D104768A1214"

// Создаем идентификатор ID-службы и его характеристику (чтение/запись)
BLEService ledService(SERVICE_ID);
BLEByteCharacteristic ledCharacteristic(CHAR_ID, BLERead | BLEWrite);
BLEDescriptor ledDescriptor("2901", "LED state");

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

   if (!BLE.begin())
   {
      Serial.println("Failed to start BLE"); // Не удалось запустить BLE
      while (1); // Останов
   }

   // Задаем название и добавляем ledService как объявляемую службу
   BLE.setLocalName("RemoteLED");
   BLE.setAdvertisedService(ledService);

   // Добавляем описание характеристики
   ledCharacteristic.addDescriptor(ledDescriptor);

   // Добавляем характеристику к службе
   ledService.addCharacteristic(ledCharacteristic);
   BLE.addService(ledService); // Добавляем службу к системе BLE
```

```
    ledCharacteristic.writeValue(0); // Начальное значение 0

    BLE.advertise();
}

void loop()
{
    BLE.poll();

    if (ledCharacteristic.written())
    {
        if (ledCharacteristic.value())
        {
            digitalWrite(LED_BUILTIN, HIGH);
        }
        else
        {
            digitalWrite(LED_BUILTIN, LOW);
        }
    }
}
```

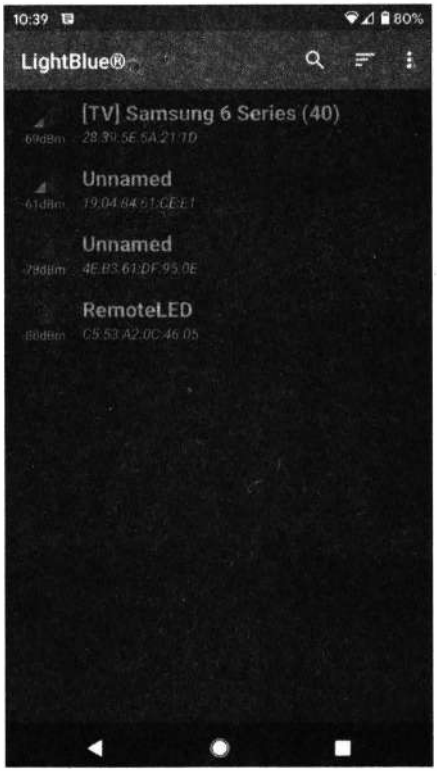
Обсуждение работы решения и возможных проблем

Скетч создает службу `ledService`, которую можно рассматривать как представление самой платы Arduino. Затем создается характеристика чтения и записи `ledCharacteristic` для представления встроенного светодиода и действий, которые можно выполнять с ним (включать и выключать). Длинные идентификаторы службы и характеристики взяты из скетча примера ButtonLED, который можно найти, выполнив команду меню **Файл | Примеры (Примеры из пользовательских библиотек) | ArduinoBLE | Peripheral | ButtonLED**. Скетч также создает описание характеристики, которое присваивает ей удобопонятное название.

В функции `setup()` скетч открывает последовательный порт, настраивает режим работы контакта, к которому подключен управляемый светодиод, и пытается запустить систему BLE. Затем устройству присваивается название `RemoteLEDS`, и служба `ledService` конфигурируется как объявляемая служба. Далее характеристика добавляется к службе, а сама служба регистрируется в системе BLE. Наконец, характеристике `ledCharacteristic` присваивается начальное значение 0 и запускается объявление службы.

В главном цикле `loop()` скетч с регулярными интервалами опрашивает систему BLE. Если в характеристику была дистанционно выполнена запись, светодиод — в зависимости от записанного значения — будет включен или выключен. Для дистанционного управления светодиодом посредством этого скетча можно использовать приложение LightBlue-Bluetooth Low Energy. Установите его на свой смартфон (Android или iOS) и подождите, пока устройство RemoteLED не отобразится в списке доступных для подключения устройств Bluetooth, как показано на рис. 14.11.

Рис. 14.11. Подключение службы RemoteLED



Если список содержит большое количество устройств Bluetooth и это усложняет работу по подключению нового устройства, нажмите значок в виде трех вертикальных точек в правой верхней части экрана и снимите галочку с опции **Show Paired Devices** (Показать подключенные устройства). Выберите в списке устройство **RemoteLED**, а затем прокрутите экран до опции **General Attributes** (Общие атрибуты) и найдите метку атрибута, заканчивающуюся текстом **a1214** с тегами **Readable** (Читаемое) и **Writable** (Записываемое). Выберите эту опцию, прокрутите экран до опции **Written Values** (Записываемые значения), установите значение 1 и нажмите кнопку **Write**. В результате подключенный к плате Arduino светодиод должен включиться. Чтобы выключить его, запишите значение 0.

Библиотека ArduinoBLE позволяет создавать устройства, реализующие многие возможности технологии BLE: датчик температуры, монитор частоты биения сердца, магнитометр и пр. С ее помощью также можно организовать подключение к периферийным устройствам с возможностями BLE и обмен данными с ними.

Дополнительная информация

Дополнительную информацию по работе с технологией Bluetooth вы найдете в книге «Make: Bluetooth», авторы Alasdair Allan, Don Coleman и Sandeep Mistry, издательство Make Community.

Дополнительную информацию по работе с устройствами Bluetooth Low Energy вы найдете в книге «Getting Started with Bluetooth Low Energy» («Технология Bluetooth Low Energy для начинающих») авторы Kevin Townsend, Carles Cufi, Akiba и Robert Davidson, издательство O'Reilly.

Дополнительная информация по службам, характеристикам и описаниям технологии Bluetooth LE приводится в спецификациях GATT для BLE (**https://oreil.ly/Qgp7S**).

ГЛАВА 15

Сети Wi-Fi и Ethernet

15.0. Введение

Хотите предоставить кому-либо еще возможность пользоваться данными с ваших датчиков? А как насчет того, чтобы позволить еще кому-нибудь управлять действиями вашего Arduino? Это может стать возможным благодаря способностям Arduino взаимодействовать с внешним миром по сетям Ethernet или Wi-Fi. В этой главе мы рассмотрим несколько способов применения платформы Arduino для работы в Интернете. Рассмотрение это основано на примерах, демонстрирующих создание и использование веб-клиентов и веб-серверов, а также применение наиболее распространенных протоколов связи Интернета для работы с платформой Arduino.

Интернет позволяет клиенту (например, веб-браузеру) запросить информацию у сервера (например, веб-сервера или поставщика услуг Интернета). Эта глава содержит как решения по созданию интернет-клиента, получающего данные от веб-службы, так и решения, в которых показывается, как Arduino может служить в качестве интернет-сервера, предоставляющего с использованием интернет-протоколов клиентам затребованную ими информацию, и даже может функционировать как веб-сервер, создающий страницы для просмотра в браузере.

Библиотеки Arduino Ethernet и Wi-Fi поддерживают широкий круг методов (стандартных интернет-протоколов), позволяющих скетчам исполнять роль интернет-клиента или сервера, и при этом скрывают от пользователя большую часть низкоуровневых подробностей.

Чтобы организовать работу клиентов и серверов и заставить их выполнять требуемые задачи, вам потребуется серьезное понимание основных принципов сетевых протоколов и адресации. Помощь в этом можно получить, обратившись к онлайновым ресурсам или к одной из следующих книг для начинающих:

- «Head First Networking» («Изучаем работу с сетями»), авторы Al Anderson и Ryan Benedetti, издательство O'Reilly.
- «Network Know-How: An Essential Guide for the Accidental Admin» («Секреты работы с сетями. Обязательное руководство для администратора-любителя»), автор John Ross, издательство No Starch Press.
- «Making Things Talk» («Чтобы вещи могли общаться»), автор Tom Igoe, издательство Make Community[1].

[1] Доступен русский перевод этой книги: Иго Т. Arduino, датчики и сети для связи устройств. 2-е изд. СПб.: «БХВ-Петербург» (**https://bhv.ru/product/arduino-datchiki-i-seti-dlya-svyazi-ustrojstv-2-e-izd/**).

Далее излагаются несколько ключевых концепций этой главы. Рекомендуем вам изучить их глубже, чем позволяют возможности этой книги, воспользовавшись предложенными дополнительными источниками информации.

Среда Ethernet

Это низкий уровень сетевой организации, предоставляющий базовую физическую возможность для передачи сообщений. Источник и назначение для этих сообщений идентифицируются адресами MAC (Media Access Control — управление доступом к среде передачи). Значение адреса MAC платы Arduino задается скетчем Arduino и должно быть однозначным в используемой локальной сети.

Среда Wi-Fi

Во многих отношениях среда Wi-Fi является функциональной заменой среды Ethernet. Подобно среде Ethernet, среда Wi-Fi предоставляет низкий уровень сетевой организации и также использует MAC-адреса для однозначной идентификации устройств в сети. Но MAC-адрес модуля Wi-Fi не требуется задавать в скетче, поскольку он присваивается модулям производителем. Как среда Wi-Fi, так и среда Ethernet находятся в нижней части стека сетевых уровней, поэтому они практически взаимозаменяемы — по крайней мере, с точки зрения программиста на языке Arduino. Организация и инициализация кода для Wi-Fi и для Ethernet слегка различаются, но после того, как подключение установится, остальной код может быть идентичным в обоих случаях.

Протоколы TCP и IP

Интернет-протоколы TCP (Transmission Control Protocol — протокол управления передачей) и IP (Internet Protocol — протокол передачи данных в Интернете) реализуются поверх уровня среды Ethernet или Wi-Fi и предоставляют возможность передачи сообщений, охватывающую весь глобальный Интернет. Доставка сообщений TCP/IP основана на системе уникальных числовых меток — IP-адресов отправителя и получателя. Сервер в Интернете также использует такую числовую метку (IP-адрес), которой не будет ни у одного другого сервера, что позволяет его однозначно идентифицировать. IP-адрес состоит из четырех байтов, обычно разделяемых точками (например, на момент подготовки этой книги IP-адрес 207.241.224.2 принадлежал веб-сайту Internet Archive). Для сопоставления текстовых идентификаторов узлов Интернета (например, **google.com**) соответствующим им числовым IP-адресам организована служба DNS (Domain Name System — служба доменных имен).

Локальные IP-адреса

При подключении к Интернету нескольких компьютеров домашней сети с помощью широкополосного маршрутизатора (шлюза) каждому из этих компьютеров маршрутизатор назначает локальный IP-адрес. Эти локальные IP-адреса присваиваются узлам сети службой DHCP (Dynamic Host Configuration Protocol — протокол

динамического конфигурирования узла), которая и дает возможность библиотекам Ethernet и Wi-Fi языка Arduino получать IP-адреса.

Протокол HTTP

Запросы веб-браузера к серверу на предоставление содержимого и соответствующие ответы сервера используют сообщения протокола HTTP (Hypertext Transfer Protocol — протокол передачи гипертекстовых данных). Чтобы веб-клиент или сервер могли должным образом реагировать, они должны понимать запросы по протоколу HTTP и отвечать на эти запросы также по протоколу HTTP. Протокол HTTP задействуется во многих решениях этой главы, разобраться с тонкостями работы которых поможет изучение упомянутых ранее руководств, в которых он рассматривается.

Язык разметки HTML

Веб-страницы обычно создаются на основе языка разметки HTML (Hypertext Markup Language — гипертекстовый язык разметки). Хотя для создания веб-сервера на Arduino использование языка HTML не является обязательным, эту возможность можно реализовать в раздаваемых таким веб-сервером веб-страницах, как и показано в решении *разд. 15.11*.

Потоковый парсинг Stream

Из-за наличия в типичной веб-странице, предназначенной для просмотра в веб-браузере, разного дополнительного текста и тегов форматирования, извлечение данных из такой страницы можно уподобить поискам иголки в стоге сена. Эту задачу можно облегчить, используя функциональность потокового парсинга Stream языка Arduino для нахождения определенных последовательностей символов и для получения строковых и числовых значений из потока данных.

Интерфейс API для веб-сайта

Весьма недальновидно и потенциально опасно создавать какую бы то ни было автоматическую систему, посылающую запросы веб-серверам, предназначенным для работы с людьми. Например, если случайно (или преднамеренно) создать код, который будет каждые 5 секунд выполнять поиск в Google, IP-адрес, с которого осуществляется исполнение этого кода, может быть заблокирован для доступа к службам Google, пока не прекратится такое поведение. В случае, например, офисной или школьной сети, в которой все устройства получают доступ к Интернету через шлюз, заблокирован будет IP-адрес этого шлюза, что лишит доступа к Интернету всех подключенных к нему устройств, создавая их пользователям чрезвычайное неудобство. По этой причине лучше всего использовать документированный интерфейс API для требуемого веб-сайта, что подробно рассмотрено в этой главе. Интерфейс API для веб-сайта позволяет получать от сервера этого сайта ответы в формате более сжатом, чем HTML, — например, в формате JSON, XML или CSV. Такой подход позволяет ограничить объем запрашиваемых данных, при этом с по-

мощью аргументов API можно сузить область запросов непосредственно до требуемых данных. Использование интерфейса API и следование его правилам позволяют работать в рамках оговоренных параметров, установленных оператором веб-сервера.

15.1. Подключение к сети Ethernet

ЗАДАЧА

Требуется подключить плату Arduino к сети Ethernet, используя для этого какой-либо модуль Ethernet — например, Ethernet-шилд или плату Ethernet FeatherWing компании Adafruit (подключается к платам с форм-фактором плат Feather компании Adafruit).

РЕШЕНИЕ

Эта задача решается с помощью скетча из листинга 15.1. Скетч запрашивает информацию с веб-сайта Internet Archive, используя библиотеку Ethernet, входящую в стандартный состав среды Arduino IDE и поддерживающую несколько модулей Ethernet для Arduino. Для правильной работы скетча в нем необходимо задать определенную информацию о вашей сети: IP-адрес сервера DNS, IP-адрес шлюза по умолчанию, а также доступный статический IP-адрес (вне диапазона IP-адресов, автоматически назначаемых сервером DHCP). Всю эту информацию можно узнать с помощью программы конфигурирования вашего сетевого маршрутизатора, которая обычно доступна через веб-браузер.

Листинг 15.1. Подключение платы Arduino к сети Ethernet

```
/*
* Скетч Ethernet Web Client
* Подключается к сети, не прибегая к использованию службы DHCP,
  а используя для устройства жестко закодированный IP-адрес
*/

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // Должен быть
                                                     // однозначный MAC-адрес
IPAddress ip(192, 168, 1, 177);       // Должен быть действительный
                                      // IP-адрес для вашей сети
char serverName[] = "archive.org";

EthernetClient client;

String request = "GET /advancedsearch.php?q=arduino&fl%5B%5D=description"
                 "&rows=1&sort%5B%5D=downloads+desc&output=csv#raw HTTP/1.0";
```

```
void setup()
{
   Serial.begin(9600);
   while(!Serial); // Для плат Leonardo и других 32-разрядных плат

   Ethernet.begin(mac, ip);
   delay(1000); // Даем оборудованию Ethernet время для инициализации

   Serial.println("Connecting to server..."); // Подключаемся к серверу
   int ret = client.connect(serverName, 80);
   if (ret == 1)
   {
      Serial.println("Connected");  // Подключились
      client.println(request);
      client.print("Host: "); client.println(serverName); // Узел:
      client.println("Connection: close"); // Подключение: закрыто
      client.println(); // Посылается завершающая пустая строка,
                        // требуемая протоколом HTTP
   }
   else
   {
      Serial.println("Connection failed, error was: ");
                     // Ошибка подключения:
      Serial.print(ret, DEC);
   }
}

void loop()
{
   if (client.available())
   {
      char c = client.read();
      Serial.print(c); // Передаем все полученные данные монитору порта
   }
   if (!client.connected())
   {
      Serial.println();
      Serial.println("Disconnecting."); // Отключаемся
      client.stop();
      while(1); // Останов
   }
}
```

Обсуждение работы решения и возможных проблем

Скетч содержит немного простого кода, позволяющего подтвердить правильность подключения и настроек вашей платы Ethernet, а также и то, что она может взаимо-

действовать с удаленными серверами. В скетче используется интерфейс API веб-сайта Internet Archive для поиска в нем материалов по Arduino, передающий в запросе параметр `q=arduino`. Переменная `request` содержит тип запроса (`GET`), его путь (`/advancedsearch.php`), а также строку запроса, в которую входит весь текст между символом `?` и пробелом перед текстом `HTTP/1.0`. После условия поиска в параметре запроса указывается только одно поле для ответа (`fl%5B%5D=description` или `fl[]=description unescaped`) и только один результат поиска (`rows=1`). Поскольку количество загрузок сортируется в нисходящем порядке (`sort%5B%5D=downloads+desc`), то этим одним возвращаемым результатом будет первый загруженный с сайта **Archive.org** ресурс по Arduino. В скетче задействован протокол HTTP 1.0, а не HTTP 1.1, поскольку серверы, работающие по протоколу HTTP 1.1, могут использовать функциональности, способные усложнить работу скетча. Например, клиенты HTTP 1.1 должны поддерживать получение ответа по частям, в результате чего сервер разбивает ответы на одну или более частей, разделенные разделителями, представляющими длину каждой части. Решение, отправлять или нет ответ по частям, принимается сервером, но если клиент в запросе указывает `HTTP/1.0`, то сервер знает, что нельзя использовать возможности протокола HTTP 1.1.

Поскольку для взаимодействия с оборудованием Ethernet скетч Arduino использует интерфейс API веб-сайта, для правильного функционирования библиотеки Ethernet является обязательным наличие в начале скетча строки кода:

```
#include <SPI.h>
```

Для того чтобы скетч мог успешно подключиться к веб-сайту и отображать результаты запроса в окне монитора порта, в нем необходимо правильно указать следующие адреса:

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

  Это MAC-адрес, который однозначно идентифицирует используемое платой Arduino оборудование Ethernet (например, Ethernet-шилд). Каждое сетевое устройство должно иметь уникальный MAC-адрес. Соответственно, в случае использования в сети нескольких Ethernet-шилдов каждый из них должен иметь свой MAC-адрес. MAC-адреса для шилдов Ethernet, выпускаемых в настоящее время, указываются на наклейке на нижней стороне платы. Если вы используете только один шилд Ethernet, то менять указанный на нем MAC-адрес нет надобности, если только по какому-то странному совпадению в вашей сети не присутствует устройство с точно таким же MAC-адресом. Использовать любой произвольный MAC-адрес для своего устройства нельзя, поскольку большинство этих адресов назначаются централизованно. Однако для адресации локальных устройств разрешается самим назначать определенные наборы MAC-адресов, при условии, что несколько устройств не используют одинаковые адреса.

  MAC-адрес состоит из последовательности байтов (называющихся *октетами*), которые в скетче представляются в виде пары полубайтов (или *ниблов*[2]), каждый

[2] Калька с англ. nibble — откусывать маленькими кусочками (или откушенный маленький кусочек), по аналогии со звучанием слова byte (бит), которое произносится так же, как и слово bite — кусать, или укус, или то, что откушено.

из которых можно представить в виде одной шестнадцатеричной цифры. В первом октете (0xDE) старший полубайт — D, а младший — E. Если младший (второй) полубайт первого октета — 2, 6, A или E, этот октет можно использовать в локальном MAC-адресе. Но, например, последовательность октетов { 0xAD, 0xDE, 0xBE, 0xEF, 0xFE, 0xED } не будет действительным локальным MAC-адресом из-за наличия во втором полубайте первого октета шестнадцатеричной цифры D, которая не входит в набор разрешенных цифр. Так что при наличии в вашей локальной сети нескольких устройств, которым нужно присвоить MAC-адреса, обязательно следуйте этому правилу, чтобы избежать неприятных сюрпризов.

- `IPAddress ip(192, 168, 1, 177);`

Это IP-адрес для идентификации устройства, осуществляющего обмен данными по сети Интернет, и он также должен быть в локальной сети уникальным. IP-адрес состоит из четырех байтов, а диапазон действительных значений каждого байта зависит от конфигурации конкретной сети. IP-адреса обычно представляются с разделителями в виде точек между значениями байтов — например: 192.168.1.177. Но во всех скетчах Arduino в качестве разделителей вместо точек используются запятые, поскольку класс `IPAddress` внутренне представляет IP-адрес как массив байтов (см. *разд. 2.4*).

Если ваша локальная сеть подключена к Интернету через маршрутизатор (роутер) или шлюз, при вызове функции `Ethernet.begin()` может потребоваться предоставить IP-адрес этого шлюза. При отсутствии этого адреса библиотека Ethernet создаст его из IP-адреса устройства, заменив последнюю группу (177 в нашем примере 192.168.1.177) значением 1. В большинстве случаев это будет правильным предположением. IP-адрес своего маршрутизатора и DNS-сервера можно узнать с помощью его конфигурационной утилиты, которая часто имеет вид веб-страницы. При необходимости в начале скетча после MAC-адреса и IP-адреса сервера добавьте IP-адреса DNS-сервера и шлюза, как показано в следующем фрагменте кода:

```
// Добавьте, если требуется для вашего маршрутизатора или шлюза
IPAddress dns_server(192, 168, 1, 2);  // IP-адрес используемого DNS-сервера
IPAddress gateway(192, 168, 1, 254);   // IP-адрес используемого шлюза
```

Также замените первую строку кода в функции `setup()` следующей — чтобы включить IP-адрес шлюза в набор значений инициализации Ethernet:

```
Ethernet.begin(mac, ip, dns_server, gateway);
```

Основная функция шлюза состоит в маршрутизации сетевых пакетов между внешней сетью (Интернетом) и локальной сетью. А DNS-сервер отвечает за преобразование текстовых названий сетевых узлов (например, **archive.org**) в числовые IP-адреса типа 207.241.224.2, чтобы предоставить библиотеке Ethernet IP-адрес веб-сервера, к которому вы пытаетесь подключиться. За кулисами библиотека Ethernet передает этот IP-адрес маршрутизатору, работу которого можно сравнить с работой почтового отделения: он погрузит ваше сообщение в «фур-

гон», который доставит его в следующее почтовое отделение на маршруте между исходной и конечной точками следования вашего сообщения. Каждое из таких почтовых отделений доставляет ваше сообщение дальше по маршруту, пока оно не достигнет своего назначения — веб-сайта **archive.org**.

Если DNS-сервер успешно сопоставляет название требуемого веб-узла его числовому IP-адресу и клиент может успешно подключиться к нему, функция `client.connect()` возвращает значение 1. Кроме значения 1 функция `client.connect()` может возвращать следующие значения:

```
 1 = success (успешное подключение)
 0 = connection failed (Подключиться не удалось)
-1 = no DNS server given (Не указан DNS-сервер)
-2 = No DNS records found (Не обнаружено записей DNS)
-3 = timeout (тайм-аут)
```

В случае значения ошибки `-1` нужно выполнить настройку DNS-сервера вручную, как описывается в решении ранее.

Большинство модулей расширения Ethernet будут работать, не требуя дополнительного конфигурирования. Но в некоторых случаях для правильной работы модуля может потребоваться указать контакт выбора устройства. В листинге 15.2 приводится выдержка из скетча примера из библиотеки Ethernet, демонстрирующая некоторые возможные значения номера этого контакта для разных модулей Ethernet.

Листинг 15.2. Значения номера контакта выбора устройства для разных модулей Ethernet

```
// Ethernet.init(10);  // Большинство шилдов для платы Arduino
// Ethernet.init(5);   // Шилд MKR ETH
// Ethernet.init(0);   // Teensy 2.0
// Ethernet.init(20);  // Teensy++ 2.0
// Ethernet.init(15);  // Плата ESP8266 с модулем
                       // Adafruit Featherwing Ethernet
// Ethernet.init(33);  // Плата ESP32 с модулем Adafruit Featherwing
```

В случае использования одного из таких модулей Ethernet вставьте соответствующую строку кода в свой скетч и расскоментируйте ее. Функцию `Ethernet.init()` необходимо вызывать первой перед вызовом функции `Ethernet.begin()`. Дополнительную информацию об этом вы найдете в документации для используемого модуля Ethernet. При успешном исполнении скетча решения в окне монитора порта должен выводиться приведенный в листинге 15.3 ответ веб-сервера **archive.org**, состоящий из заголовков HTTP, за которым следует пустая строка, а затем тело ответа. Листинг также содержит диагностическую информацию, создаваемую скетчем и указывающую на инициацию процесса подключения клиента к серверу, успешное подключение, а затем отключение.

Листинг 15.3. Ответ веб-сервера archive.org на запрос, отправленный скетчем из листинга 15.2

```
Connecting to server... (Подключаемся к серверу...)
Connected               (Подключились)
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 24 Nov 2019 03:36:50 GMTы
Content-Type: text/csv;charset=UTF-8
Connection: close
Content-disposition: attachment; filename=search.csv
Strict-Transport-Security: max-age=15724800

"description"
"Arduino The Documentary 2010"

Disconnecting.          (Отключаемся)
```

Дополнительная информация

Подробная информация по библиотеке Ethernet Arduino приведена на веб-странице **https://oreil.ly/JPlKM** веб-сайта Arduino.

15.2. Получение IP-адреса устройства автоматически

ЗАДАЧА

Требуется автоматически получать для устройства Ethernet (например, шилда) уникальный IP-адрес от DHCP-сервера.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 15.4. Процесс конфигурации в этом скетче похож на процесс из скетча решения, приведенного в *разд. 15.1*, но в нем методу (функции) `Ethernet.begin()` не передается IP-адрес модуля Ethernet.

Листинг 15.4. Автоматическое получение IP-адреса от сервера DHCP

```
/*
* Скетч DHCP
* Получает IP-адрес от DHCP-сервера и отображает его в окне монитора порта
*/

#include <SPI.h>
#include <Ethernet.h>
```

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // Должен быть
                                                     // уникальный MAC-адрес

EthernetClient client;

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Для плат Leonardo и других 32-разрядных плат

   if (Ethernet.begin(mac) == 0)
   {
      Serial.println("Failed to configure Ethernet using DHCP");
                  // Не удалось настроить Ethernet по DHCP
      while(1); // Останов
   }
   delay(1000); // Даем оборудованию Ethernet время для инициализации

}

#define MAINTAIN_DELAY 750 // Обновляем аренду DHCP каждые 0,75 секудны

void loop()
{
   static unsigned long nextMaintain = millis() + MAINTAIN_DELAY;
   if (millis() > nextMaintain)
   {
      nextMaintain = millis() + MAINTAIN_DELAY;
      int ret = Ethernet.maintain();
      if (ret == 1 || ret == 3)
      {
         Serial.print("Failed to maintain DHCP lease. Error: ");
                   // Не удалось обновить аренду DHCP. Ошибка:
         Serial.println(ret);
      }

      Serial.print("Current IP address: "); // Текущий IP-адрес
      IPAddress myIPAddress = Ethernet.localIP();
      Serial.println(myIPAddress);
   }
}
```

Обсуждение работы решения и возможных проблем

Основное отличие этого скетча от скетча решения из *разд. 15.1* состоит в том, что в нем не используется явный IP-адрес устройства (или шлюза и DNS-сервера). Значения этих адресов скетч получает от сервера DHCP в начале своего исполнения. Скетч также содержит проверку на успешное выполнение оператора `Ethernet.begin()`. Эта проверка необходима для того, чтобы убедиться в получении действи-

тельного IP-адреса от сервера DHCP (без действительного IP-адреса доступ к сети невозможен).

Сервер DHCP присваивает IP-адрес модулю Ethernet на определенное время, называющееся *арендой DHCP*. По истечении аренды сервер DHCP присваивает модулю новый IP-адрес, который может быть таким же, как и прежний, но также может быть и другим. Функцию `Ethernet.maintain()` необходимо вызывать периодически, чтобы сервер DHCP знал, что модуль Ethernet все еще является активным. Если эту функцию не вызывать по крайней мере один раз в секунду, по истечении предыдущей аренды DHCP она может не обновиться. Поведение аренды DHCP (длительность аренды, действия сервера DHCH при обновлении аренды и т. п.) зависит от конфигурации используемого сетевого маршрутизатора. Но при каждом обновлении аренды DHCP этот скетч отображает полученный IP-адрес в окне монитора порта.

Использование возможностей DHCP увеличивает размер скетча на пару килобайт, поэтому при ограниченном объеме памяти лучше все же использовать фиксированный IP-адрес (см. решение в *разд. 15.1*).

15.3. Обмен простыми сообщениями по протоколу UDP

ЗАДАЧА

Требуется реализовать обмен простыми сообщениями через Интернет.

РЕШЕНИЕ

Эта задача решается с помощью скетча из листинга 15.5. В скетче используется библиотека Arduino UDP (User Datagram Protocol — пользовательский протокол данных) для обмена строковыми сообщениями. По сравнению с протоколом TCP протокол UDP более простой, но чуть менее аккуратный протокол для обмена сообщениями. Например, если по назначению прибудут не все пакеты сообщения TCP, это вызовет сообщение об ошибке, тогда как пакеты сообщений UDP могут прибывать по назначению в порядке ином, чем они были отправлены, или не прибыть вообще, и отправляющий скетч не получит никакого сообщения об ошибке. Однако у протокола UDP меньшие накладные расходы, чем у протокола TCP, что делает его хорошим выбором, когда можно пожертвовать надежностью доставки сообщений ради более высокой скорости их доставки. В этом простом примере скетч Arduino отображает полученную строку в окне монитора порта и посылает обратно отправителю строку `"acknowledged"`, подтверждая прием исходной строки.

Листинг 15.5. Скетч для обмена сообщениями по протоколу UDP

```
/*
* Скетч UDPSendReceiveStrings
* Принимает строковые сообщения по протоколу UDP, отображает их
  в окне монитора порта и посылает отправителю строку "acknowledged"
*/
```

```
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // Уникальный MAC-адрес
unsigned int localPort = 8888; // Локальный порт, на котором ожидаются UDP-сообщения

// Буферы для входящих и исходящих данных
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; // Буфер для хранения входящих пакетов,
char replyBuffer[] = "acknowledged";       // Строка, посылаемая обратно отправителю

// Экземпляр объекта UDP для обмена сообщениями по протоколу UPD
EthernetUDP Udp;

void setup()
{
   Serial.begin(9600);
   // Инициализируем Ethernet и UDP
   Ethernet.begin(mac);
   Udp.begin(localPort);
}

void loop()
{
   // Если есть доступные данные, считываем пакет
   int packetSize = Udp.parsePacket();
   if(packetSize)
   {
      Serial.print("Received packet of size "); // Получен пакет размером
      Serial.println(packetSize);
      // Считываем пакет в переменную packetBuffer и получаем IP-адрес
      // и номер порта отправителя
      Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
      Serial.println("Contents:"); // Содержимое:
      Serial.println(packetBuffer);
      // Посылаем строку обратно отправителю
      Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
      Udp.write(replyBuffer);
      Udp.endPacket();
   }
   maintainLease(); // Удерживаем аренду DHCP
   delay(10);
}

#define MAINTAIN_DELAY 750 // Обновляем аренду DHCP каждые 0,75 секунды

void maintainLease()
{
```

```
  static unsigned long nextMaintain = millis() + MAINTAIN_DELAY;
  if (millis() > nextMaintain)
  {
    nextMaintain = millis() + MAINTAIN_DELAY;
    int ret = Ethernet.maintain();
    if (ret == 1 || ret == 3)
    {
      Serial.print("Failed to maintain DHCP lease. Error: ");
                // Не удалось обновить аренду DHCP. Ошибка:
      Serial.println(ret);
    }
    Serial.print("Current IP address: "); // Текущий IP-адрес
    IPAddress myIPAddress = Ethernet.localIP();
    Serial.println(myIPAddress);
  }
}
```

Вы можете проверить это, запустив на своем компьютере скетч из листинга 15.6. Этот скетч Processing отправляет скетчу Arduino сообщения по протоколу UDP (подробно о среде разработки Processing рассказано во *врезке «Среда разработки Processing» в главе 4*). В скетче используется библиотека UDP разработки Stephane Cousot. Установить эту библиотеку можно, выполнив команду меню среды Processing **Эскиз | Импортировать библиотеку | Добавить библиотеку**, выбрав в списке библиотеку UDP и нажав кнопку **Установить**. При исполнении скетча Arduino в окне монитора порта будет отображаться текущий IP-адрес его шилда Ethernet. Адрес IP, указанный в строке:

```
String ip = "192.168.1.177";
```

скетча Processing нужно заменить этим адресом.

Листинг 15.6. Скетч Processing для отправки сообщения UDP скетчу Arduino

```
// Посылает скетчу Arduino сообщения и принимает ответы по протоколу UDP
// Нажмите любую клавишу, чтобы послать сообщение "Hello Arduino" скетчу Arduino
import hypermedia.net.*; // Импортируем библиотеку UDP
                         // (разработчик Stephane Cousot) для Processing

UDP udp; // Определяем объект UDP

void setup()
{
  udp = new UDP( this, 6000 ); // Создаем подключение UDP на порту 6000
  //udp.log( true );           // <-- отображаем действия подключения
  udp.listen( true );          // и ожидаем входящие сообщения
}
```

```
void draw()
{
   // Здесь ничего не делаем
}

void keyPressed()
{
   String ip = "192.168.1.177"; // IP-адрес назначения
   int port = 8888; // Порт назначения
   udp.send("Hello World", ip, port ); // Отправляемое сообщение
}

void receive( byte[] data )
{
   for(int i=0; i < data.length; i++)
   print(char(data[i]));
   println();
}
```

Обсуждение работы решения и возможных проблем

Вставьте шилд Ethernet в плату Arduino и подключите его к разъему сетевой платы компьютера (или к свободному порту маршрутизатора или сетевого коммутатора вашей домашней сети). Загрузите скетч Arduino в плату Arduino и запустите на компьютере скетч Processing. Щелкните мышью на окне приложения Processing, сделав его активным. Теперь нажмите любую клавишу, чтобы послать сообщение `"Hello Arduino"` скетчу Arduino. Скетч Arduino отображает полученное им сообщение в окне монитора порта и посылает скетчу Processing строку `"acknowledged"`, которая отображается в панели консоли окна Processing. Длина строки сообщения ограничена значением постоянной, определенной в заголовочном файле EthernetUdp.h. Значение по умолчанию этой константы равно 24, но его можно увеличить, отредактировав соответствующим образом в этом файле следующую строку:

```
#define UDP_TX_PACKET_MAX_SIZE 24
```

Протокол UDP предоставляет простой и быстрый способ обмениваться сообщениями в сети Ethernet. Но у него есть свои ограничения: нет гарантии доставки сообщений, и в сети с большой загрузкой некоторые сообщения могут быть утеряны или доставлены не в том порядке, в котором они были отправлены. Однако протокол UDP хорошо подходит для передачи таких сообщений, как значения состояний датчиков, — каждое сообщение содержит текущее значение датчика, и любое утерянное сообщение заменяется следующими сообщениями.

Передача и прием сообщений со значениями датчиков показаны в скетче Arduino из листинга 15.7. Скетч принимает сообщения, содержащие значения, которые должны быть записаны в порты аналогового вывода, и отвечает отправителю со значениями на выводах аналогового ввода.

Листинг 15.7. Скетч Arduino для приема и передачи значений датчиков по протоколу UDP

```
/*
*Скетч UDPSendReceive
*/

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUDP.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // Уникальный MAC-адрес

unsigned int localPort = 8888; // Локальный порт, на котором ожидаются UDP-сообщения

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; // Буфер для хранения входящих пакетов
int packetSize;                            // Переменная для хранения объема пакетов

const int analogOutPins[] = { 3,5,6,9 };

// Экземпляр объекта UDP для обмена сообщениями по протоколу UPD
EthernetUDP Udp;

void setup()
{
   Ethernet.begin(mac,ip);
   Udp.begin(localPort);

   Serial.begin(9600);
   Serial.println("Ready");
}

void loop()
{
   // Если есть доступные данные, считываем пакет
   packetSize = Udp.parsePacket();
   if(packetSize > 0)
   {
      Serial.print("Received packet of size ");
                // Получен пакет размером в
      Serial.print(packetSize);
      Serial.println(" with contents:"); // с содержимым
      // Считываем пакет в переменную packetBuffer и получаем IP-адрес
      // и номер порта отправителя
      packetSize = min(packetSize,UDP_TX_PACKET_MAX_SIZE);
      Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
```

```
      for( int i=0; i < packetSize; i++)
      {
         byte value = packetBuffer[i];
         if( i < 4)
         {
            // Записываем только в первые четыре выходных аналоговых контакта
            analogWrite(analogOutPins[i], value);
         }
         Serial.println(value, DEC);
      }
      Serial.println();
      // Сообщаем отправителю значения на аналоговых контактах
      sendAnalogValues(Udp.remoteIP(), Udp.remotePort());
   }
   // Берем небольшую паузу
   delay(10);
}

void sendAnalogValues( IPAddress targetIp, unsigned int targetPort )
{
   int index = 0;
   for(int i=0; i < 6; i++)
   {
      int value = analogRead(i);

      packetBuffer[index++] = lowByte(value);  // Младший байт
      packetBuffer[index++] = highByte(value); // Старший байт
   }
   // Посылаем сообщение обратно отправителю
   Udp.beginPacket(targetIp, targetPort);
   Udp.write(packetBuffer);
   Udp.endPacket();
}
```

Скетч посылает и принимает значения на аналоговых портах с 0 по 5, используя двоичные данные (подробно обмен сообщениями в двоичном формате описан во *введении* в *главу 4* (*разд. 4.0*), а также в *разд. 4.6* и *4.7*).

Разница между передачей данных по последовательному интерфейсу и протоколу UDP состоит лишь в том, что в последнем случае вместо оператора `Serial.write()` используется оператор `Udp.write()`.

В листинге 15.8 приводится скетч Processing, работающий совместно со скетчем Arduino из листинга 15.7. Скетч создает шесть ползунков, расположенных по центру окна приложения Processing. Эти ползунки можно перетаскивать мышью, задавая таким образом уровни для записи функцией `analogWrite()` на шесть аналоговых выходных контактов платы Arduino. Полученные обратно от платы Arduino данные

выводятся в окно консоли окна Processing. Установив ползунки, нажмите любую клавишу на клавиатуре, чтобы отправить их значения скетчу Arduino.

Листинг 15.8. Скетч Processing для обмена данными со скетчем Arduino по протоколу UDP

```
// Скетч Processing UDPTest
// Отправляет скетчу Arduino данные и получает ответы, используя протокол UDP

import hypermedia.net.*;

UDP udp; // Определяем объект UDP

HScrollbar[] scroll = new HScrollbar[6]; // См: topics/gui/scrollbar

void setup()
{
   size(256, 200);
   noStroke();
   for (int i=0; i < 6; i++)     // Создаем ползунки
   scroll[i] = new HScrollbar(0, 10 + (height / 6) * i, width, 10, 3*5+1);
   udp = new UDP( this, 6000 ); // Создаем подключение UDP на порте 6000
   udp.listen( true );          // и ожидаем входящие сообщения
}

void draw()
{
   background(255);
   fill(255);
   for (int i=0; i < 6; i++)
   {
      scroll[i].update();
      scroll[i].display();
   }
}

void keyPressed()
{
   String ip = "192.168.137.64"; // IP-адрес шилда Ethernet
        // платы Arduino (ОТКОРРЕКТИРУЙТЕ ИХ ДОЛЖНЫМ ОБРАЗОМ!)
   int port = 8888; // Порт назначения
   byte[] message = new byte[6] ;
   for (int i=0; i < 6; i++)
   {
      message[i] = byte(scroll[i].getPos());
      println(int(message[i]));
   }
```

```
   println();
   udp.send( message, ip, port );
}

void receive( byte[] data )
{
   println("incoming data is:"); // Входящие данные:
   for (int i=0; i < min(6, data.length); i++)
   {
      scroll[i].setPos(data[i]);
      print(i);
      print(":");
      println((int)data[i]);
   }
}

class HScrollbar
{
   int swidth, sheight;  // Ширина и высота полосы
   int xpos, ypos;       // Позиции X и Y полосы
   float spos, newspos;  // Позиция X ползунка
   int sposMin, sposMax; // Минимальное и максимальное значения ползунка
   int loose;            //
   Boolean over;         // Курсор мыши над ползунком?
   Boolean locked;
   float ratio;

   HScrollbar (int xp, int yp, int sw, int sh, int l)
   {
      swidth = sw;
      sheight = sh;
      int widthtoheight = sw - sh;
      ratio = (float)sw / (float) widthtoheight;
      xpos = xp;
      ypos = yp-sheight/2;
      spos = xpos + swidth/2 - sheight/2;
      newspos = spos;
      sposMin = xpos;
      sposMax = xpos + swidth - sheight;
      loose = l;
   }

   void update()
   {
      if (over())
      {
         over = true;
      }
```

```
    else
    {
        over = false;
    }
    if (mousePressed && over)
    {
        locked = true;
    }
    if (!mousePressed)
    {
        locked = false;
    }
    if (locked)
    {
        newspos = constrain(mouseX-sheight/2, sposMin, sposMax);
    }
    if (abs(newspos - spos) > 1)
    {
        spos = spos + (newspos-spos)/loose;
    }
}

int constrain(int val, int minv, int maxv)
{
    return min(max(val, minv), maxv);
}

 Boolean over()
{
    if (mouseX > xpos && mouseX < xpos+swidth &&
    mouseY > ypos && mouseY < ypos+sheight)
    {
        return true;
    }
    else
    {
        return false;
    }
}

void display()
{
    fill(255);
    rect(xpos, ypos, swidth, sheight);
    if (over || locked)
    {
        fill(153, 102, 0);
    }
```

```
      else
      {
         fill(102, 102, 102);
      }
      rect(spos, ypos, sheight, sheight);
   }

   float getPos()
   {
      return spos * ratio;
   }

   void setPos(int value)
   {
      spos = value / ratio;
   }
}
```

15.4. Использование платы Arduino со встроенным модулем Wi-Fi

ЗАДАЧА

Требуется создавать проекты беспроводных сетей с помощью платы Arduino, оснащенной встроенным модулем Wi-Fi.

РЕШЕНИЕ

Некоторые платы Arduino кроме основного микроконтроллера ARM или AVR также содержат отдельный встроенный микроконтроллер, обеспечивающий функциональность Wi-Fi. Такие платы последних выпусков основаны на модуле NINA-W102 компании u-blox, который, в свою очередь, построен на микросхеме ESP32 компании Espressif.

Скетч решения приводится в листинге 15.9. В скетче используется библиотека WiFiNINA, которая устанавливается с помощью Менеджера библиотек среды Arduino IDE. Эта библиотека поддерживает модуль Wi-Fi, встроенный в платы Arduino Uno WiFi Rev 2, Nano 33 IoT, MKR 1010 и MKR VIDOR 4000. Скетч также совместим с модулями Airlift компании Adafruit — такими как адаптерная плата (артикул 4201) и шилд (артикул 4285), но сама компания рекомендует использовать свою специализированную библиотеку WiFiNINA (**https://oreil.ly/OSay1**).

Чтобы подключиться к сети Wi-Fi, вставьте в указанных местах в скетче идентификатор (SSID) и пароль вашей сети.

Листинг 15.9. Скетч для работы в сетях Wi-Fi с платами, оснащенными встроенным модулем Wi-Fi

```
/*
* Скетч WiFiNINA Web Client
* Запрашивает данные с веб-сайта Internet Archive
*/

#include <SPI.h>
#include <WiFiNINA.h>

const char ssid[] = "ВАШ_SSID";         // Идентификатор (SSID) вашей сети Wi-Fi
const char password[] = "ВАШ_ПАРОЛЬ";  // Пароль вашей сети Wi-Fi

WiFiClient client; // Клиент Wi-Fi

char serverName[] = "archive.org";
String request = "GET /advancedsearch.php?q=arduino&fl%5B%5D=description"
                    "&rows=1&sort%5B%5D=downloads+desc&output=csv#raw HTTP/1.0";

bool configureNetwork()
{
   int status = WL_IDLE_STATUS; // Статус Wi-Fi

   if (WiFi.status() == WL_NO_MODULE)
   {
      Serial.println("Couldn't find WiFi hardware.");
                  // Не удалось обнаружить оборудование Wi-Fi
      return false;
   }
   String fv = WiFi.firmwareVersion();
   if (fv < WIFI_FIRMWARE_LATEST_VERSION)
   {
      Serial.println("Please upgrade the WiFi firmware");
                  // Необходимо обновить прошивку Wi-Fi
   }
   while (status != WL_CONNECTED)
   {
      Serial.print("Attempting WiFi connection to ");
                // Пытаемся подключиться к сети Wi-Fi
      Serial.println(ssid);
      status = WiFi.begin(ssid, password); // Продолжаем попытки
                                           // подключения, пока не подключимся
      delay(1000); // Пауза длительностью в 1 секунду
   }
   return true;
}
```

```
void setup()
{
   Serial.begin(9600);
   if (!configureNetwork())
   {

      Serial.println("Stopping.");
      while(1); // halt
   }
   Serial.println("Connecting to server..."); // Подключаемся к серверу
   int ret = client.connect(serverName, 80);
   if (ret == 1)
   {
      Serial.println("Connected"); // Подключились
      client.println(request);
      client.print("Host: "); client.println(serverName);
      client.println("Connection: close");
      client.println();
   }
   else
   {
      Serial.println("Connection failed, error was: ");
                  // // Ошибка подключения:
      Serial.print(ret, DEC);
   }
}

void loop()
{
   if (client.available())
   {
      char c = client.read();
      Serial.print(c); // Передаем все полученные данные монитору порта
   }
   if (!client.connected())
   {
      Serial.println();
      Serial.println("Disconnecting."); // Отключаемся
      client.stop();
      while(1); // Останов
   }
}
```

Если требуется подключиться к SSL-серверу, то вместо клиента `WiFiClient` нужно использовать клиент `WiFiSSLClient` и при вызове функции `client.connect()` подключаться к SSL-порту сервера (обычно это порт 443) вместо порта 80.

Обсуждение работы решения и возможных проблем

Скетч этого решения очень похож на скетч решения из *разд. 15.1*, но с несколькими важными изменениями (подробная информация о структуре запроса и HTTP-протоколе приводится в *разд. «Обсуждение работы решения и возможных проблем»* того решения). Самым важным и очевидным различием является использование сети Wi-Fi вместо сети Ethernet для подключения к веб-серверу. А также, поскольку код для конфигурирования модуля Wi-Fi чуть более сложный, чем для Ethernet, он собран в отдельную функцию `configureNetwork()`. За исключением этих различий остальной код в функции `setup()` и главном цикле `loop()` такой же, что и для скетча с использованием сети Ethernet.

В скетче задействуется не заданный фиксированный IP-адрес, а адрес, выделяемый службой DHCP. В случае работы с шилдом Ethernet привлечение службы DHCP значительно увеличивает размер скетча, но при использовании библиотеки WiFiNINA такого увеличения объема не происходит, поскольку большая часть работы выполняется микроконтроллером модуля Wi-Fi. При желании с библиотекой WiFiNINA можно использовать и фиксированный IP-адрес. Для этого перед вызовом функции `WiFi.begin()` нужно с помощью функции `IPAddress ip()` объявить требуемый IP-адрес (например: `IPAddress ip(192, 168, 0, 177);`), а затем вызвать функцию `WiFi.config(ip)`. Все платы Arduino, оснащенные функциональностью Wi-Fi, уже имеют MAC-адрес, который присваивается их Wi-Fi-модулю при изготовлении, потому определять MAC-адрес в скетче не нужно.

Для правильной работы этого скетча в среде Arduino IDE необходимо установить поддержку для используемой платы, а также библиотеку WiFiNINA. Процесс установки поддержки платы следующий. В среде Arduino IDE выполните команду **Инструменты | Плата | Менеджер плат**. В открывшемся окне Менеджера плат для платы Arduino Uno WiFi Rev 2 выберите опцию **Arduino megaAVR Boards** и нажмите кнопку **Установить**. Для плат Nano 33 IoT, MKR WiFi 1010 или MKR Vidor 4000 таким же способом установите поддержку **Arduino SAMD Boards**. И для всех плат установите с помощью Менеджера библиотек библиотеку WiFiNINA. Установив библиотеку WiFiNINA и поддержку для используемой платы, подключите плату и выберите ее и используемый ею порт, выполнив команды меню **Инструменты | Плата** и **Инструменты | Порт** соответственно.

Откройте окно монитора порта — если все работает должным образом, плата должна подключиться к вашей сети Wi-Fi и в окне монитора порта должно отобразиться сообщение, показанное в листинге 15.10.

Листинг 15.10. Сообщение, выводимое в окне монитора порта при успешном подключении к сети Wi-Fi

```
Please upgrade the WiFi firmware
(Необходимо обновить прошивку Wi-Fi)
Attempting WiFi connection to YOUR_SSID
(Пытаемся подключиться к вашей сети Wi-Fi)
Connecting to server...
(Подключаемся к серверу...)
```

```
Connected
(Подключились)
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Sun, 24 Nov 2019 02:46:19 GMT
Content-Type: text/csv;charset=UTF-8
Connection: close
Content-disposition: attachment; filename=search.csv
Strict-Transport-Security: max-age=15724800

"description"
"Arduino The Documentary 2010"

Disconnecting.
(Отключаемся)
```

Наличие в сообщении извещения о необходимости обновления прошивки Wi-Fi означает, что микропрограммное обеспечение модуля Wi-Fi вашей платы устарело и требует обновления. Чтобы обновить его, выполните команду меню среды Arduino IDE **Инструменты | WiFi101/WiFiNINA Firmware Updater**. В открывшемся диалоговом окне сначала выберите используемую вами плату, откройте скетч обновления (**Open Updater sketch**) и загрузите его в плату Arduino. Затем выберите самую последнюю версию микропрограммного обеспечения для своей платы и нажмите кнопку **Update Firmware**.

15.5. Подключение к сети Wi-Fi с помощью недорогих модулей

ЗАДАЧА

Требуется создавать бюджетные проекты со встроенными возможностями Wi-Fi, используя платформу Arduino.

РЕШЕНИЕ

Платформа Arduino поддерживает сетевые проекты — такие как, например, веб-серверы и веб-клиенты. Проекты таких беспроводных сетей можно реализовывать с помощью платы Arduino, оснащенной возможностью Wi-Fi. На рынке предлагается большой выбор модулей Wi-Fi для Arduino и их клонов, но из всех этих модулей только один стоит меньше двух долларов (при оптовых закупках) — это модуль ESP-01 на основе микросхемы ESP8266 компании Espressif Systems. В отличие от большинства модулей Wi-Fi, модуль ESP-01 не имеет встроенной поддержки USB, но эту проблему можно решить с помощью адаптера USB/RS-322 (см. *разд.* «*Аппаратные средства для последовательной связи*» *главы 4*) или использовать в качестве такого адаптера плату Arduino, как показано далее в этом разделе.

Создайте в среде Arduino IDE пустой скетч, выполнив команду меню **Файл | Примеры | 01.Basics | Bare Minimum**, а затем подключите модуль ESP-01 к плате Arduino (рис. 15.1). Для питания этого модуля требуется напряжение 3,3 В, но взять это напряжение с контакта 3V3 платы Arduino не получится, поскольку он не может предоставить ток достаточной силы, требуемый для модуля. Поэтому питание нужно брать с контакта 5V, предварительно подав его на стабилизатор напряжения с выходным напряжением 3,3 В — например, на LD1117V33. Подключение этого стабилизатора напряжения также показано на рис. 15.1.

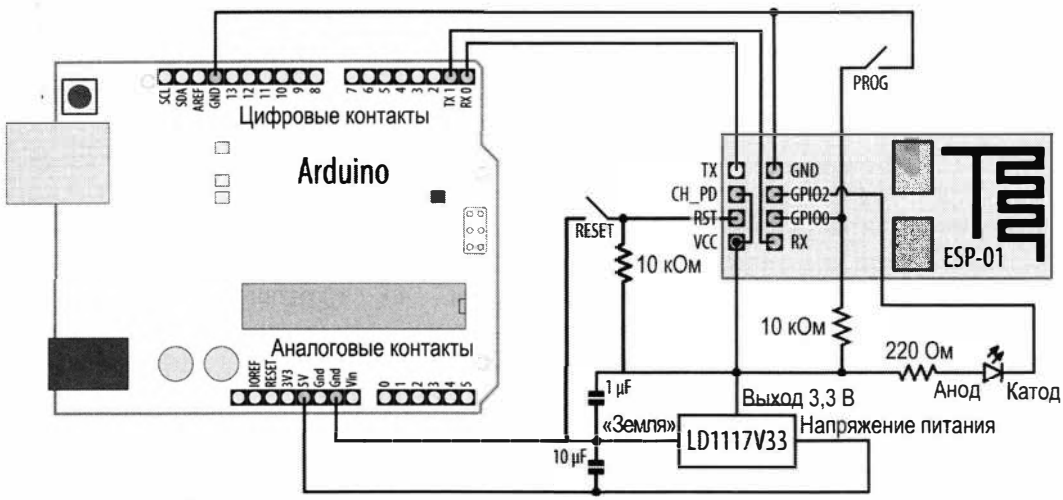


Рис. 15.1. Использование платы Arduino в качестве адаптера USB/RS-232 для модуля ESP-01

Можно также использовать модуль Wi-Fi со встроенной поддержкой USB — например, плату Feather HUZZAH с модулем ESP8266 компании Adafruit (артикул 2821) или плату SparkFun ESP8266 Thing Dev Board компании SparkFun (артикул WRL-1371). Для этих плат также потребуется установить поддержку модуля ESP8266, но использовать плату Arduino в качестве адаптера USB/RS-232, как показано на рис. 15.1, тогда будет не нужно. Эти платы стоят дороже, чем простой модуль ESP-01, но они предоставляют невероятное удобство работы. И вам не нужно будет подключать кнопки PROG и RESET, как это требуется для модуля ESP-01, поскольку процесс сброса этих плат осуществляется встроенным в них загрузчиком.

Прежде чем программировать модуль ESP8266, необходимо в среде Arduino IDE установить его поддержку. Для этого выполните команду меню **Файл | Настройки** и щелкните на значке справа от поля ввода **Дополнительные ссылки для менеджера плат**. В открывшемся одноименном окне вставьте в новую строку ссылку:

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

нажмите кнопку **OK**, а затем закройте диалоговое окно, нажав в нем кнопку **OK**. После чего откройте окно Менеджера плат (**Инструменты | Платы | Менеджер плат**) и выполните поиск по ESP8266. В результатах поиска выберите и установите пакет esp8266 by ESP8266 Community.

Далее выполните команду меню **Инструменты | Плата** и в разделе **ESP8266** списка плат выберите **Generic ESP8266 Module**, потом откройте список портов (**Инструменты | Порт**) и в списке портов COM укажите порт платы Arduino, к которой подключен модуль ESP8266. Затем выполните команду меню **Инструменты | Builtin Led** и выберите в списке значение 2. Хотя модуль ESP8266 и оснащен встроенным светодиодом, использование его может нарушить работу последовательного интерфейса. Поэтому мы применим внешний светодиод, подключенный к контакту GPIO2 (General Purpose Input-Output — ввод/вывод общего назначения) платы (см. рис. 15.1). Осталось откорректировать скетч, приведенный в листинге 15.11, указав имя SSID и пароль своей сети Wi-Fi.

Нажмите на модуле ESP-01 кнопку PROG и, удерживая ее нажатой, загрузите в него откорректированный скетч, при этом наблюдая за панелью консоли в нижней части окна среды Arduino IDE. Когда в этой панели отобразится сообщение **Connecting**, нажмите и отпустите кнопку RESET на модуле ESP-01, продолжая удерживать нажатой кнопку PROG. Возможно, этот процесс нужно будет повторить несколько раз, чтобы начать загрузку скетча в модуль. Индикатором успешного запуска процесса загрузки будет отображение в панели консоли среды Arduino IDE сообщения типа **Writing at 0x00000000... (7 %)**, за которым будут следовать другие сообщения. Может быть, эти сообщения не будут прокручиваться в панели, поэтому удерживайте кнопку PROG нажатой до тех пор, пока в ней не отобразится сообщение **Done uploading**. После этого нажмите кнопку RESET, чтобы перезапустить модуль. Откройте монитор порта — в нем должен выводиться URL-адрес веб-страницы, которую можно посетить, введя этот адрес в адресную строку браузера. При каждом нажатии кнопки на этой веб-странице будет мигать подключенный к модулю светодиод. После загрузки программы в модуль ESP-01 его можно отключить от платы Arduino или адаптера USB/RS-232 и подать на него питание 3,3 В (ни в коем случае не больше). Модуль будет исполнять загруженную программу до тех пор, пока на него будет подаваться питание.

Листинг 15.11. Скетч для управления светодиодом, подключенным к модулю ESP-01

```
/*
* Скетч ESP-01
* Управляет светодиодом с веб-страницы
*/

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

const char* ssid = "ВАШ_SSID";        // Название (SDID) вашей сети Wi-Fi
const char* password = "ВАШ_ПАРОЛЬ"; // Пароль вашей сети Wi-Fi

ESP8266WebServer server(80);

const int led = LED_BUILTIN;
int ledState = LOW;
```

```
// Создаем форму HTML с кнопкой
static const char formText[] PROGMEM =
"<form action=\"/\">\n"
"<input type=\"hidden\" name=\"toggle\"/>\n"
"<button type=\"submit\">Toggle LED</button>\n"
"</form>\n";

// Обрабатываем запросы на предоставление корневого (/) документа
void handleRoot()
{
   // Если сервер получил аргумент toggle, переключаем
   // состояние светодиода
   if (server.hasArg("toggle"))
   {
      ledState = !ledState;
      digitalWrite(led, !ledState);
   }

   // Display the form
   server.send(200, "text/html", FPSTR(formText));
}

// Сообщение об ошибке для запросов неизвестных файлов
void handleNotFound()
{
   server.send(404, "text/plain", "File not found\n\n"); // Файл не найден
}

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

   pinMode(led, OUTPUT);
   digitalWrite(led, !ledState);

   // Initialize WiFi
   WiFi.mode(WIFI_STA);
   WiFi.begin(ssid, password);

   // Ожидаем установления подключения
   while (WiFi.status() != WL_CONNECTED)
   {
      delay(500);
      Serial.print(".");
   }
   Serial.println();
```

```
    // Организовываем обработчик для запроса корневого документа (/)
    // и всего прочего
    server.on("/", handleRoot);
    server.onNotFound(handleNotFound);

    server.begin(); // Запускаем сервер
}

#define MSG_DELAY 10000
void loop()
{
    static unsigned long nextMsgTime = 0;

    server.handleClient(); // Обрабатываем запросы от клиентов HTTP

    if (millis() > nextMsgTime)
    {
        // Отображаем адрес URL в окне монитора порта
        Serial.print("Visit me at http://"); // Посетите страницу http://
        Serial.println(WiFi.localIP());
        nextMsgTime = millis() + MSG_DELAY;
    }
}
```

Обсуждение работы решения и возможных проблем

Скетч этого решения содержит заголовки, требуемые для подключения к сети Wi-Fi и создания веб-сервера. В нем определяются переменные для названия (SSID) и пароля вашей сети Wi-Fi (которые нужно будет указать в обозначенных местах скетча, прежде чем загружать его в модуль ESP-01), а также объект для веб-сервера. В нем также создаются переменные для хранения номера контакта светодиода (`LED_BUILTIN`), его состояния (`ledState`) и фрагмента кода HTML, который отображает кнопку (`formText`). Последняя переменная сохраняется во флеш-памяти, а не в оперативной энергозависимой памяти модуля (см. решение в *разд. 17.3*). Наличие постоянного подключения к «земле» контакта GPIO2 модуля ESP-01 не позволит должным образом выполняться его загрузке, поэтому для управления светодиодом используется обратная логика. Поскольку светодиод подключен к линии 3,3 В, подача сигнала низкого уровня на контакт GPIO2 вызовет включение светодиода.

Пример, показанный в скетче решения, немного более сложный, чем примеры для сети Ethernet или для плат со встроенной возможностью Wi-Fi, которые мы рассматривали ранее. Это объясняется более широкими возможностями библиотеки ESP8266WebServer по сравнению со стандартной библиотекой Server среды Arduino IDE. В частности, эта библиотека содержит ряд функций, которые упрощают процесс предоставления веб-документов. Эти задачи можно реализовать и с другими платами, но библиотеки, входящие в состав пакета поддержки плат ESP8266, намного упрощают их.

За константой HTML-кода веб-формы следуют два объявления функций-*обработчиков*. Задача обработчика заключается в обработке запросов ресурсов — например, веб-страницы. Все веб-серверы имеют корневой документ. Например, запрашивая веб-сайт **http://oreilly.com** с завершающей косой чертой (/) или без нее, мы запрашиваем корневой документ (/) этого сайта. Обработчик запроса корневого документа `handleRoot` сначала проверяет наличие в запросе параметра `toggle`. В случае положительного результата проверки он меняет состояние сигнала управления светодиодом на обратное — с высокого (`HIGH`) на низкое (`LOW`) и наоборот. После этого он отображает фрагмент HTML-кода, содержащийся в константе `formText`.

Фрагмент HTML-кода создает форму, которая ассоциируется с действием (/, корневой документ веб-сервера). При подаче формы она передает обработчику в виде аргументов все элементы ввода. Форма содержит скрытый элемент `toggle` и кнопку, исполняющую роль активатора подачи (`type="submit"`). По нажатии кнопки форма посылает значение `toggle` в виде аргумента, что вызывает изменение состояния светодиода на обратное.

Следующий обработчик посылает код ошибки 404 при запросе у сервера любого другого ресурса. Затем следует функция `setup()`, в которой инициализируется объект `Serial`, задается выходной (`OUTPUT`) режим работы контакта `led` управления светодиодом, инициализируется объект `WiFi` и осуществляется подключение к сети Wi-Fi. После чего выполняется конфигурирование сервера определенными ранее обработчиками и осуществляется запуск сервера.

В главном цикле `loop()` вызывается метод (функция) сервера `server.handleClient()` для обработки поступающих запросов. Кроме этого, каждые 10 секунд в окно монитора порта выводится адрес URL и предложение посетить веб-страницу по этому адресу. То есть в любое время можно открыть монитор порта и посмотреть этот адрес. Откройте веб-страницу с формой, расположенную по этому адресу, в браузере, работающем в той же самой сети Wi-Fi, что и модуль ESP-01, и испытайте работу кнопки.

На момент подготовки этой книги производитель модуля ESP8266 (**https://espressif.com/**) предлагал 16 разных его версий, различающихся объемом памяти, количеством контактов и размерами. На вики-странице сообщества ESP8266 (**https://oreil.ly/bK8-w**) содержится хороший их обзор.

Чтобы позволить использование этих модулей в проектах IoT (Internet of Things — Интернета вещей), ряд поставщиков компонентов для нужд сообщества мейкеров[3] разработали платы на основе этого модуля, содержащие возможности USB и другие функциональности, которые дают возможность заряжать батарею и осуществлять простое программирование при подключении к среде Arduino IDE. Самый простой способ начать работать с возможностями модуля ESP8266 — это приобрести одну из таких плат, оснащенную USB-интерфейсом, например плату Feather HUZZAH компании Adafruit (артикул 2821) или плату ESP8266 Thing компании

[3] Кто такие мейкеры, см., например, здесь: **https://the-steppe.com/razvitie/kto-takie-meykery-otvechaem-na-populyarnye-voprosy-svyazannye-s-molodoy-subkulturoy**.

SparkFun (артикул WRL-13231). Обе эти платы сравнительно недорогие, но все же дороже простых модулей на основе ESP8266, таких как ESP-01. Далее в *разд. «Дополнительная информация»* приводятся ссылки на пошаговые учебные пособия, помогающие начать работу с обеими этими платами.

Все такие модули имеют более чем достаточный объем памяти и вычислительную мощность, чтобы использовать их в большинстве проектов. Модуль ESP-01 оснащен 32-разрядным микропроцессором с рабочей частотой по умолчанию 80 МГц. Он несет на борту 80 Кбайт памяти RAM и — в зависимости от версии — от 512 Кбайт до 16 Мбайт флеш-памяти.

Дополнительная информация

Руководство по подключению платы ESP8266 Thing компании SparkFun приведено на странице **https://oreil.ly/IAzBN**.

Руководство по подключению платы ESP8266 Feather Huzzah компании Adafruit приведено на странице **https://oreil.ly/woknW**.

По адресу **https://oreil.ly/ZgaaE** приведена статья в журнале Make Magazine, поясняющая процесс подключения модуля ESP-01.

Модуль ESP32 представляет собой значительно улучшенный вариант модуля ESP-01. Он имеет больший объем памяти, более высокую рабочую частоту, а также обеспечивает поддержку технологии Bluetooth LE. Его можно использовать как самостоятельную микроконтроллерную плату, но он также применяется в качестве модуля Wi-Fi в таких платах, как Arduino MKR WiFi 1010 и Arduino Uno WiFi Rev2 (см. решение в *разд. 15.4*).

15.6. Извлечение данных из ответа веб-сервера

ЗАДАЧА

Требуется получить данные от веб-сервера. Например, извлечь из ответа веб-сервера данные разных типов (строковые, целочисленные, с плавающей запятой).

РЕШЕНИЕ

Скетч решения приводится в листинге 15.12. Скетч обращается к веб-службе Open Notify (**http://open-notify.org/Open-Notify-API**), представляющей данные о местонахождении международной космической станции (МКС). Из ответа сервера службы скетч извлекает время ответа и местонахождение МКС в значениях широты и долготы и отображает полученные данные в окне монитора порта. Скетч может работать с платами, поддерживающими библиотеки Ethernet или WiFiNINA, а также с платами на основе микроконтроллера ESP8266. Для указания типа платы следует раскомментировать соответствующий оператор `#include` в начале скетча. Скетч состоит из четырех файлов. Основной файл скетча приводится в листинге 15.12, а в листингах 15.13–15.15 приведены заголовочные файлы для каждого типа платы.

Для работы скетча сначала необходимо установить библиотеку TimeLib (подробно об этой библиотеке рассказано в *разд. 12.4*).

Листинг 15.12. Основной скетч для извлечения данных из ответа веб-сервера

```
/*
* Скетч Client-agnostic web data extraction
* Скетч может работать с платами ESP8266, WiFiNINA и шилдами Ethernet
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"     // Для плат WiFiNINA
//#include "USE_Ethernet.h" // Для плат Ethernet
//#include "USE_ESP8266.h"  // Для шилдов ESP8266

#include <TimeLib.h>

char server[] = "api.open-notify.org";

void setup()
{
   Serial.begin(9600);
   if (!configureNetwork()) // Start the network
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      {
         delay(0); // Останов. Платы на ESP8266 не любят бесконечных
                  // циклов без задержки
      }
   }
   int ret = client.connect(server, 80);

   if (ret == 1)
   {
      Serial.println("Connected"); // Подключились
      client.println("GET /iss-now.json HTTP/1.0"); // Запрос HTTP
      client.print("Host: "); client.println(server);
      client.println("Connection: close"); // Подключение закрыто
      client.println();
   }
   else
   {
      Serial.println("Connection failed, error was: ");
                  // // Ошибка подключения:
      Serial.print(ret, DEC);
```

```
      while(1)
      {
         delay(0); // Останов. Платы на ESP8266 не любят бесконечных
                   // циклов без задержки
      }
   }
}

char timestampMarker[] = "\"timestamp\":";
char posMarker[] = "\"iss_position\":";

void loop()
{
   if (client.available())
   {
      if (client.find('"')) // Начало идентификатора строки
      {
         String id = client.readStringUntil('"');
         if (id.equals("timestamp")) // Начало временной метки
         {
            if (client.find(':')) // После каждого идентификатора
                                  // следует двоеточие (☺
            {
               unsigned long timestamp = client.parseInt();
               setTime(timestamp); // Устанавливаем часы на время ответа
               digitalClockDisplay();
            }
            else
            {
               Serial.println("Failed to parse timestamp.");
                           // Не удалось извлечь данные из временной метки
            }
         }
         if (id.equals("iss_position")) // Начало данных местоположения
         {
            if (client.find(':')) // После каждого идентификатора
                                  // следует двоеточие (:)
            {
               // Метки начинаются с двойной кавычки ("), а данные
               // местоположения заканчиваются закрывающей
               // фигурной скобкой (})
               while (client.peek() != '}' && client.find('"'))
               {
                  // Считываем метку
                  String id = client.readStringUntil('"');
                  float val = client.parseFloat(); // Считываем значение
                  client.find('"'); // Находим концевую двойную кавычку (")
                                    // после значения с плавающей запятой
```

```
                Serial.print(id + ": "); Serial.println(val, 4);
            }
        }
        else
        {
            Serial.println("Failed to parse position data.");
                        // Не удалось извлечь данные местоположения
        }
      }
    }
    if (!client.connected())
    {
      Serial.println();
      Serial.println("disconnecting."); // Отключаемся
      client.stop();
      while(1)
      {
            delay(0); // Останов. Платы на ESP8266 не любят бесконечных
                      // циклов без задержки
      }
    }
}

String padDigits(int digit)
{
    String str = String("0") + digit;       // Вставляем 0 перед цифрой
    return str.substring(str.length() - 2); // Удаляем все символы,
                                            // кроме двух последних
}

void digitalClockDisplay()
{
    String datestr = String(year()) + "-" + padDigits(month()) +
                                      "-" + padDigits(day());
    String timestr = String(hour()) + ":" + padDigits(minute()) +
                                      ":" + padDigits(second());
    Serial.println(datestr + " " + timestr);
}
```

В листинге 15.13 приводится исходный код заголовочного файла для плат на микроконтроллере ESP8266. Тогда как основному скетчу можно дать любое название, заголовочным файлам необходимо присвоить конкретные имена. Для создания заголовочного файла щелкните мышью на значке направленного вниз треугольника в правом конце панели вкладок среды Arduino IDE (находится непосредственно под значком монитора порта) и в открывшемся меню выберите пункт **Новая вкладка**.

В появившемся поле **Имя нового файла** введите название USE_ESP8266.h и нажмите кнопку **OK** — в панели вкладок появится новая вкладка с этим названием и откроется окно нового файла. Скопируйте в него содержимое листинга 15.13 и сохраните файл. При этом обратите внимание на необходимость заменить идентификаторы ВАШ_SSID и ВАШ_ПАРОЛЬ на имя (SSID) и пароль для вашей сети Wi-Fi.

Листинг 15.13. Заголовочный файл для плат на микроконтроллере ESP8266

```
#include <SPI.h>
#include <ESP8266WiFi.h>

const char ssid[] = "ВАШ_SSID";          // SSID вашей сети Wi-Fi
const char password[] = "ВАШ_ПАРОЛЬ";    // Пароль вашей сети Wi-Fi

WiFiClient client;

bool configureNetwork()
{
   WiFi.mode(WIFI_STA);
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) // Ожидаем установления
                                         // подключения
   {
      delay(1000);
      Serial.print("Waiting for connection to ");
                // Ожидаем подключения к
      Serial.println(ssid);
   }
   return true;
}
```

В листинге 15.14 приводится исходный код заголовочного файла для варианта на шилде Ethernet. Этот файл создается так же, как и заголовочный файл для варианта на микроконтроллере ESP8266, но назвать его нужно USE_Ethernet.h. Если вы предпочитаете использовать конкретный IP-адрес, обратитесь к решению из *разд. 15.1* и откорректируйте код этого заголовочного файла соответствующим образом.

Листинг 15.14. Заголовочный файл для варианта плат на шилде Ethernet

```
#include <SPI.h>
#include <Ethernet.h>

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

EthernetClient client;
```

```
bool configureNetwork()
{
   if (Ethernet.begin(mac))
   {
      delay(1000); // Даем оборудованию Ethernet время для инициализации
      return true;
   }
   else
   {
      return false;
   }
}
```

В отличие от решения из *разд. 15.2*, в этом решении нет вызова функции `Ethernet.maintain()` для обновления аренды DHCP. Ни для плат WiFiNINA, ни для плат на микроконтроллере ESP8266 вызывать периодически функцию `maintain()` для обновления аренды DHCP не требуется. Но для проектов с использованием шилдов Ethernet, исполняющихся в течение длительного времени, обновление аренды DHCP необходимо. Пример использования в скетче функции `maintain()` можно посмотреть в решении из *разд. 15.8*.

В листинге 15.15 приводится исходный код заголовочного файла для варианта на плате WiFiNINA. Этот файл создается так же, как и заголовочный файл для варианта на микроконтроллере ESP8266, но назвать его нужно `USE_NINA.h`. При этом обратите внимание на необходимость заменить идентификаторы `ВАШ_SSID` и `ВАШ_ПАРОЛЬ` на имя (SSID) и пароль для вашей сети Wi-Fi.

Листинг 15.15. Заголовочный файл для варианта на плате WiFiNINA

```
#include <SPI.h>
#include <WiFiNINA.h>

const char ssid[] = "ВАШ_SSID";          // SSID вашей сети Wi-Fi
const char password[] = "ВАШ_ПАРОЛЬ";    // Пароль вашей сети Wi-Fi

WiFiClient client;

bool configureNetwork()
{
   int status = WL_IDLE_STATUS; // Статус Wi-Fi
   if (WiFi.status() == WL_NO_MODULE)
   {
      Serial.println("Couldn't find WiFi hardware.");
                  // Не удалось обнаружить оборудование Wi-Fi.
      return false;
   }
```

```
    String fv = WiFi.firmwareVersion();
    if (fv < WIFI_FIRMWARE_LATEST_VERSION)
    {
      Serial.println("Please upgrade the WiFi firmware");
                  // Необходимо обновить прошивку Wi-Fi
    }
    while (status != WL_CONNECTED)
    {
      Serial.print("Attempting WiFi connection to ");
                // Пытаемся подключиться к сети Wi-Fi
      Serial.println(ssid);
      status = WiFi.begin(ssid, password); // Продолжаем попытки
                            // подключения, пока не подключимся
      delay(1000); // Пауза длительностью в 1 секунду
    }
    return true;
}
```

Обсуждение работы решения и возможных проблем

Веб-служба API ISS веб-сайта Open Notify возвращает результаты запроса в формате JSON (JavaScrip Object Notation — система обозначений объектов JavaScript), состоящем из пар «атрибут — значение» в виде `"атрибут": значение`. Скетч посылает запрос веб-серверу таким же способом, как и в решениях из *разд. 15.1* и *15.4*.

Поиск значений в полученном ответе JSON осуществляется с использованием функциональности потокового парсинга Stream, с которой мы познакомились в *разд. 4.5*. В цикле `loop()` выполняется поиск символа двойных кавычек ("), который обозначает начало метки, — например, временной метки. Обнаружив метку атрибута временной метки, скетч извлекает первое следующее за меткой целое число, представляющее собой количество секунд, прошедших с начала UNIX-времени. Удобно, что эти значения совместимы с функциями из библиотеки Time, с которыми мы работали в решении из *разд. 12.4*, поэтому скетч может использовать эти функции, чтобы задать текущее время. Затем скетч выводит на экран время, используя функцию `ClockDisplay()`, похожую на соответствующую функцию из того решения.

При обнаружении идентификатора `iss_position` скетч ищет еще две метки, которыми будут `latitude` (широта) и `longitude` (долгота), извлекает их соответствующие значения с плавающей запятой и отображает их в окне монитора порта. При отсутствии других меток (эти две метки должны быть последними) или обнаружении символа закрывающей фигурной скобки (который обозначает конец идентификатора `iss_position`), скетч завершает обработку ответа. Далее приводится пример ответа веб-сервиса, в котором значения атрибутов выделены полужирным шрифтом:

```
{"message": "success", "timestamp": 1574635904, "iss_position":
{"latitude": "-37.7549", "longitude": "95.5304"}}
```

В результате обработки этого ответа в окне монитора порта должна отобразиться следующая информация:

```
Connected

2019-11-24 22:51:44
latitude: -37.7549
longitude: 95.5304

disconnecting.
```

Дополнительная информация

Документация по интерфейсу API веб-службы Open Notify приведена на веб-странице **https://oreil.ly/OaTHs**.

На веб-странице **https://oreil.ly/_3OTa** вы найдете список веб-интерфейсов API, доступных для широкой публики.

15.7. Запрос данных у веб-сервера, использующего формат XML

ЗАДАЧА

Требуется получать данные с веб-сайта, который предоставляет их в формате XML. Например, получить значения определенных полей ответа, выдаваемого сайтом погоды с помощью API-интерфейса в формате XML.

РЕШЕНИЕ

Скетч для решения этой задачи приведен в листинге 15.16. Скетч получает информацию о погоде в Лондоне от веб-сайта службы Open Weather. Подобно скетчу из предыдущего решения, для этого скетча также требуется создать один из таких же трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую вашему сетевому интерфейсу. В случае использования плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующий заголовочный файл необходимо вписать правильное имя (SSID) и пароль вашей беспроводной сети.

Листинг 15.16. Скетч для получения информации от веб-службы в формате XML

```
/*
 * Скетч Simple Weather Client
 * Запрашивает и получает данные с веб-сайта http://openweathermap.org/
 * Считывает значение температуры из поля temperature value
 * Записывает полученное значение в выходной аналоговый порт
```

```
// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"     // Для плат WiFiNINA
//#include "USE_Ethernet.h" // Для шилдов Ethernet
//#include "USE_ESP8266.h"  // Для плат ESP8266

char serverName[] = "api.openweathermap.org";

String request =
       "GET /data/2.5/weather?q=London,UK&units=imperial&mode=xml&APPID=";
String APIkey = "YOUR_KEY_HERE"; // См. пояснение в тексте

void setup()
{
   Serial.begin(9600);
   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      delay(0); // Останов. Платы на ESP8266 не любят бесконечных
                // циклов без задержки
   }
}

void loop()
{
   if (client.connect(serverName, 80) > 0)
   {
      Serial.println("Connected"); // Подключились
      // get weather
      client.println(request + APIkey + " HTTP/1.0");
      client.print("Host: "); client.println(serverName);
      client.println("Connection: close");
      client.println();
   }
   else
   {
      Serial.println(" connection failed"); // Ошибка подключения
   }
   if (client.connected())
   {
      if (client.find("<temperature value=") )
      {
         int temperature = client.parseInt();
         Serial.print("Temperature: "); Serial.println(temperature);
      }
```

```
      else
         Serial.print("Could not find temperature field");
                  // Не удалось найти поле температуры
      if (client.find("<humidity value="))
      {
         int humidity = client.parseInt();
         Serial.print("Humidity: "); Serial.println(humidity);
      }
      else
         Serial.print("Could not find humidity field");
                  // Не удалось найти поле влажности
   }
   else
   {
      Serial.println("Disconnected");
   }
   client.stop();
   client.flush();
   delay(60000); // Выжидаем одну минуту перед следующим обновлением
}
```

Обсуждение работы решения и возможных проблем

Веб-служба Open Weather (**https://oreil.ly/dLliX**) предоставляет погодные данные для свыше 200 тысяч населенных пунктов по всему миру. Служба бесплатная для нечастого использования, но требует регистрации, чтобы получить ключ API. Условия использования службы и информация по получению ключа приводятся на веб-странице **https://oreil.ly/p-IRB**.

Скетч подключается к веб-сайту **api.openweathermap.org**, а затем отправляет веб-службе по адресу **http://api.openweathermap.org/data/2.5/weather** следующий запрос:

```
?q=London,UK&units=imperial&mode=xml&APPID= YOUR_KEY_HERE
```

Строка, следующая за `q=`, указывает населенный пункт и страну (полный список поддерживаемых населенных пунктов находится здесь: **https://oreil.ly/oXLrI**). Настройка `units=imperial` задает возвращение значения температуры в градусах Фаренгейта, а `mode=xml` — возвращение запрошенных данных в формате XML. В строке кода:

```
String API key = "YOUR_KEY_HERE";
```

идентификатор `YOUR_KEY_HERE` (`ВАШ_КЛЮЧ_ЗДЕСЬ`) нужно заменить на выданный вам ключ API службы Open Weather Map.

Возвращенные XML данные выглядят, как показано в листинге 15.17.

Листинг 15.17. Данные, возвращаемые службой Open Weather Map

```
<current>
   <city id="2643743" name="London">
      <coord lon="-0.13" lat="51.51"/>
      <country>GB</country>
      <timezone>0</timezone>
      <sun rise="2019-11-25T07:34:34" set="2019-11-25T16:00:36"/>
   </city>
   <temperature value="48" min="45" max="51.01" unit="fahrenheit"/>
   <humidity value="93" unit="%"/>
   <pressure value="1004" unit="hPa"/>
   <wind>
      <speed value="6.93" unit="mph" name="Light breeze"/>
      <gusts/>
      <direction value="100" code="E" name="East"/>
   </wind>
   <clouds value="75" name="broken clouds"/>
   <visibility value="10000"/>
   <precipitation mode="no"/>
   <weather number="803" value="broken clouds" icon="04n"/>
   <lastupdate value="2019-11-25T02:02:46"/>
</current>
```

Скетч использует функцию `client.find()`, чтобы найти в возвращенном ответе теги `temperature` и `humidity`, а затем с помощью функции `client.parseInt()` извлекает значения этих параметров. Скетч посылает запрос на получение погодных данных каждые 60 секунд. В рассматриваемом случае мы имеем дело с XML-сообщением сравнительно небольшого размера. Но обработка XML-сообщений большого размера может потребовать большего объема памяти и вычислительной мощности, чем способна предоставить плата Arduino. Поэтому может быть более предпочтительно запрашивать данные в более компактном формате JSON, как показано в решении из *разд. 15.6*.

15.8. Организация веб-сервера на платформе Arduino

ЗАДАЧА

Требуется организовать на основе платформы Arduino раздачу веб-страниц с помощью веб-сервера. Например, чтобы просматривать в браузере значения датчиков, подключенных к аналоговым контактам платы Arduino.

РЕШЕНИЕ

Такая задача решается с помощью скетча из листинга 15.18. Этот скетч основан на скетче примера Ethernet WebServer, входящего в стандартный состав среды Arduino

IDE, который считывает и отображает значения входных аналоговых контактов платы Arduino. Исходный скетч был модифицирован для работы с платами, оснащенными модулями WiFiNINA (см. *разд. 15.4*), а также с платами на основе микроконтроллера ESP8266 (см. *разд. 15.5*). Требуемый тип платы указывается раскомментированием соответствующего оператора `#include` в начале скетча. Скетч состоит из четырех файлов. Основной файл скетча приводится в листинге 15.18, а в листингах 15.19–15.21 приведены заголовочные файлы для каждого типа платы.

Листинг 15.18. Основной скетч веб-сервера на платформе Arduino

```
/*
* Скетч Web Server
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для шилдов Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

void setup()
{
   Serial.begin(9600);
   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      delay(0); // Останов. Платы на ESP8266 не любят бесконечных циклов без задержки
   }
   server.begin();
}

#define MSG_DELAY 10000
void loop()
{
   static unsigned long nextMsgTime = 0;
   if (millis() > nextMsgTime)
   {
      Serial.print("Visit me at http://");  // Откройте веб-страницу http://
      Serial.println(getIP());
      nextMsgTime = millis() + MSG_DELAY;
   }

   maintain(); // При необходимости обновляем аренду DHCP

   client = server.available(); // Ожидаем подключения
   if (client)
```

```
{
   Serial.println("New client connection"); // Подключился новый клиент

   // Запрос HTTP завершается пустой строкой
   boolean currentLineIsBlank = true;
   while (client.connected())
   {
      if (client.available())
      {
         char c = client.read();
         Serial.write(c);
         // Если достигли конца пустой строки и обнаружили другой
         // символ новой строки \n, тогда достигнут конец заголовков
         if (c == '\n' && currentLineIsBlank)
         {
            // Посылаем стандартный заголовок ответа HTTP:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println("Connection: close"); // Закрываем
                                                 // подключение после отправки ответа
            client.println("Refresh: 5"); // Обновляем каждые 5 секунд
            client.println(); // Конец заголовков

            client.println("<!DOCTYPE HTML>");
            client.println("<HTML>");

            // Отображаем значение каждого входного
            // аналогового контакта
            for (int analogChannel = 0; analogChannel < 6; analogChannel++)
            {
               int sensorReading = analogRead(analogChannel);
               client.print("A"); client.print(analogChannel);
               client.print(" = "); client.print(sensorReading);
               client.println("<BR />");
            }
            client.println("</HTML>");
            break; // Принудительный выход из цикла while
         }
         if (c == '\n')
         {
            // Начинаем новую строку
            currentLineIsBlank = true;
         }
         else if (c != '\r')
         {
            // Получили символ в текущей строке
            currentLineIsBlank = false;
         }
      }
   }
```

```
        // Даем браузеру время, чтобы получить данные:
        delay(100);

        // Закрываем подключение
        client.stop();
        Serial.println("Client disonnected"); // Клиент отключен
    }
}
```

В листинге 15.19 приводится исходный код заголовочного файла для варианта на шилде Ethernet. Тогда как основному скетчу можно дать любое название, заголовочным файлам необходимо присвоить конкретные имена. Для создания заголовочного файла щелкните мышью на значке направленного вниз треугольника в правом конце панели вкладок среды Arduino IDE (находится непосредственно под значком монитора порта) и в открывшемся меню выберите пункт **Новая вкладка**. В появившемся поле **Имя нового файла** введите название `USE_Ethernet.h` и нажмите кнопку **OK** — в панели вкладок появится новая вкладка с этим названием и откроется окно нового файла. Скопируйте в него содержимое листинга 15.19 и сохраните файл.

Поскольку основной скетч исполняется в течение длительного времени, этот заголовочный файл содержит функцию `maintain()`, которая вызывается в главном цикле `loop()` основного скетча для обновления аренды DHCP (см. *разд. 15.2*). Если вы предпочитаете использовать конкретный IP-адрес, обратитесь к решению из *разд. 15.1* и откорректируйте код этого заголовочного файла соответствующим образом.

Листинг 15.19. Заголовочный файл для решения веб-сервера с использованием шилда Ethernet

```
#include <SPI.h>
#include <Ethernet.h>

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

EthernetClient client;
EthernetServer server(80);

bool configureNetwork()
{
   if (Ethernet.begin(mac))
   {
      delay(1000); // Даем оборудованию Ethernet время для инициализации
      return true;
   }
   else
   {
      return false;
   }
}
```

```
IPAddress getIP()
{
   return Ethernet.localIP();
}
#define MAINTAIN_DELAY 750 // Обновляем аренду DHCP каждые 0,75 секунды

void maintain()
{
   static unsigned long nextMaintain = millis() + MAINTAIN_DELAY;
   if (millis() > nextMaintain)
   {
      nextMaintain = millis() + MAINTAIN_DELAY;
      int ret = Ethernet.maintain();
      if (ret == 1 || ret == 3)
      {
         Serial.print("Failed to maintain DHCP lease. Error: ");
                  // Не удалось обновить аренду DHCP. Ошибка:
         Serial.println(ret);
      }
   }
}
```

В листинге 15.20 приводится исходный код заголовочного файла для плат на микроконтроллере ESP8266. Этот файл создается так же, как и заголовочный файл для варианта на шилде Ethernet, но назвать его нужно `USE_ESP8266.h`.

При этом обратите внимание на необходимость заменить идентификаторы `ВАШ_SSID` и `ВАШ_ПАРОЛЬ` на имя (SSID) и пароль для вашей сети Wi-Fi. Функция `maintain()` не содержит никакого рабочего кода, поскольку при использовании плат на основе микроконтроллера ESP8266 (так же как и при варианте на платах WiFiNINA) обновлять вручную аренду DHCP не требуется.

Листинг 15.20. Заголовочный файл для решения веб-сервера с использованием плат на основе микроконтроллера ESP8266

```
#include <SPI.h>
#include <ESP8266WiFi.h>

const char ssid[] = "ВАШ_SSID";
const char password[] = "ВАШ_ПАРОЛЬ";

WiFiClient client;
WiFiServer server(80);

bool configureNetwork()
{
   WiFi.mode(WIFI_STA);
   WiFi.begin(ssid, password);
```

```
   while (WiFi.status() != WL_CONNECTED) // Ожидаем установления подключения
   {
      delay(1000);
      Serial.print("Waiting for connection to ");
      Serial.println(ssid);
   }
   return true;
}

IPAddress getIP()
{
   return WiFi.localIP();
}

void maintain()
{
   // Ничего не делаем
}
```

В листинге 15.21 приводится исходный код заголовочного файла для варианта на плате WiFiNINA. Этот файл создается так же, как и заголовочный файл для варианта на микроконтроллере ESP8266, но назвать его нужно USE_NINA.h. При этом обратите внимание на необходимость заменить идентификаторы ВАШ_SSID и ВАШ_ПАРОЛЬ на имя (SSID) и пароль для вашей сети Wi-Fi.

Листинг 15.21. Заголовочный файл для решения веб-сервера на плате WiFiNINA

```
#include <SPI.h>
#include <WiFiNINA.h>

const char ssid[] = "ВАШ_SSID";
const char password[] = "ВАШ_ПАРОЛЬ";

WiFiClient client;
WiFiServer server(80);

bool configureNetwork()
{
   int status = WL_IDLE_STATUS; // Статус Wi-Fi

   if (WiFi.status() == WL_NO_MODULE)
   {
      Serial.println("Couldn't find WiFi hardware.");
                  // Не удалось обнаружить оборудование Wi-Fi
      return false;
   }
```

```
    String fv = WiFi.firmwareVersion();
    if (fv < WIFI_FIRMWARE_LATEST_VERSION)
    {
       Serial.println("Please upgrade the WiFi firmware");
                  // Необходимо обновить прошивку Wi-Fi
    }
    while (status != WL_CONNECTED)
    {
       Serial.print("Attempting WiFi connection to ");
                // Пытаемся подключиться к сети Wi-Fi
       Serial.println(ssid);
       status = WiFi.begin(ssid, password); // Продолжаем попытки
                              // подключения, пока не подключимся
       delay(1000); // Пауза длительностью в 1 секунду
    }
    return true;
}

IPAddress getIP()
{
    return WiFi.localIP();
}

    void maintain()
{
    // Ничего не делаем
}
```

Обсуждение работы решения и возможных проблем

Подобно скетчу решения из *разд. 15.6*, этот скетч использует один из трех заголовочных файлов для подключения к сети. Например, если подключить к основному скетчу заголовочный файл `USE_Ethernet.h`, подключение будет осуществляться к сети Ethernet с использованием службы DHCP для получения IP-адреса. Поскольку при использовании службы DHCP в среде Ethernet требуется вручную обновлять аренду DHCP, этот заголовочный файл содержит определение функции `maintain()`, которая вызывается в цикле `loop()` скетча. Это определение не требовалось в заголовочном файле для варианта на шилде Ethernet в решении из *разд. 15.6*, поскольку код этого скетча исполняется только один раз с последующим остановом. В варианте с платой на микроконтроллере ESP8266 в скетче нужно подключить заголовочный файл `USE_ESP8266.h`, а при использовании платы WiFiNINA — файл `USE_NINA.h`. В каждом их этих трех заголовочных файлов определяется переменная клиента и сервера и предоставляются методы для конфигурирования сети (`configureNetwork()`) и получения присвоенного IP-адреса (`getIP()`).

В начале исполнения скетч запускает последовательный порт, затем выполняет конфигурирование сети, используя функцию, соответствующую используемому

сетевому оборудованию, после чего запускает сервер. В главном цикле `loop()` каждые 10 секунд отображается сообщение, содержащее URL-адрес, по которому можно подключаться к серверу. При вводе в адресную строку браузера этого URL-адреса, должна открыться веб-страница, содержащая значения на выходных аналоговых контактах 0 по 6 платы Arduino (тема аналоговых контактов рассматривается более подробно в *главе 6*).

В функции `setup()` две строки кода инициализируют библиотеку Ethernet и выполняют конфигурирование веб-сервера предоставленным IP-адресом. Главный цикл `loop()` ждет, а затем обрабатывает запросы от клиента:

```
client = server.available();
```

В этом коде объект `client` представляет подключенного к веб-серверу клиента. В зависимости от используемого заголовочного файла это будет объект `EthernetClient` или `WiFiClient`.

Оператор условия `if(client)` выполняет проверку на успешное подключение клиента.

В операторе цикла `while(client.connected())` выполняется проверка подключения веб-сервера к клиенту, подающему запрос.

Функция `client.available()` проверяет наличие данных от клиента, а функция `client.read()` считывает байт доступных данных. Этот процесс похож на использование функций `Serial.available()` и `Serial.read()`, рассматриваемое в *главе 4*, с той лишь разницей, что данные поступают из Интернета, а не из последовательного порта. Код считывает данные до тех пор, пока не будет обнаружена первая строка без данных, что обозначает конец запроса. В ответ на запрос посредством функции `client.println()` посылаются заголовки HTTP, а затем отображаются выходные значения аналоговых контактов платы Arduino.

Пакет поддержки плат на микроконтроллере ESP8266 содержит большой набор библиотек для создания веб-серверов. Если только вы не разрабатываете код для поддержки большого разнообразия сетевого оборудования, вам может оказаться полезным использовать эти возможности. Более подробная информация на эту тему приведена в *разд. 15.5*.

15.9. Обработка входящих запросов от веб-клиентов

ЗАДАЧА

Требуется управлять выходными сигналами на цифровых и аналоговых контактах платы Arduino, играющей роль веб-сервера. Например, управлять уровнями на определенных контактах на основании параметров, посылаемых из браузера.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 15.22. Скетч читает полученный от браузера запрос и изменяет значения цифровых и аналоговых контактов согласно пара-

метрам запроса. Адрес URL для подключения к скетчу выводится в окно монитора порта.

Запрос содержит одно или несколько полей, идентификаторы которых состоят из слова `pin`, за которым следует буква типа контакта: `D` (digital, цифровой) или `A` (analog, аналоговый), а затем номер контакта. После идентификатора поля следует знак равенства (=), а затем значение поля.

Например, отправка запроса `http://IP_ADDRESS/?pinD2=1` из адресной строки браузера устанавливает высокий логический уровень (значение 1) на цифровом контакте 2, а отправка запроса `http://IP_ADDRESS/?pinD2=0` устанавливает на этом контакте низкий логический уровень (значение 0). Идентификатор *IP_ADDRESS* в строке запроса нужно заменить на IP-адрес, который отображается в окне монитора порта. Результаты обработки запроса отображаются в окне монитора порта, а также могут отображаться изменением состояния светодиодов, подключенных к соответствующим контактам платы Arduino (о подключении светодиодов к контактам платы Arduino рассказано в *главе 7*).

Точно так же, как и для скетча из предыдущего решения, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующем заголовочном файле необходимо указать правильное имя (SSID) и пароль используемой беспроводной сети. Плата ESP8266 имеет ограниченное количество выходных контактов, узнать которые можно, изучив документацию на используемую плату. Подключение светодиодов к некоторым контактам может вызвать непредсказуемое поведение платы.

Результаты обработки веб-сервером запроса, отправленного ему из адресной строки браузера, показаны на рис. 15.2.

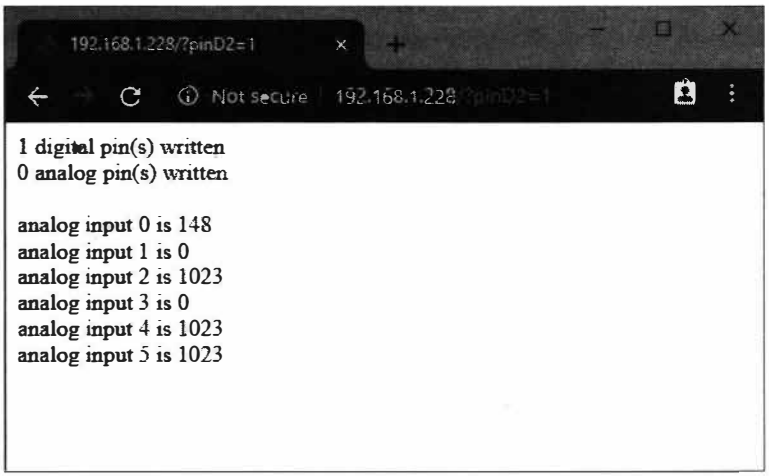


Рис. 15.2. Отображение в окне браузера результата обработки веб-сервером полученного запроса

Листинг 15.22. Скетч веб-сервера для обработки запросов веб-клиента

```
/*
* Скетч Incoming request
* Отвечает на отправляемые в URL-адресе запросы для изменения
  состояния выходных цифровых и аналоговых контактов платы Arduino
* Отображает в браузере количество портов с измененными состояниями
  и входные значения аналоговых портов
* Например:
* Запрос http://IP_ADDRESS/?pinD2=1 устанавливает высокий логический
  уровень (значение 1) на контакте 2
* Запрос http://IP_ADDRESS/?pinD2=0 устанавливает низкий логический
  уровень (значение 0) на контакте 2
* В скетче также демонстрируется извлечение значений из текста
  с помощью класса Stream среды Arduino IDE
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для шилдов Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

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

   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      delay(0); // halt; ESP8266 does not like ∞ loop without a delay
                // Останов. Платы на ESP8266 не любят бесконечных
                // циклов без задержки
   }
   server.begin();
}

#define MSG_DELAY 10000
void loop()
{
   static unsigned long nextMsgTime = 0;
   if (millis() > nextMsgTime)
   {
      Serial.print("Try http://"); // Попробуйте ввести этот URL-адрес
      Serial.print(getIP()); Serial.println("?pinD2=1");
      nextMsgTime = millis() + MSG_DELAY;
   }
```

```
maintain(); // При необходимости вручную обновляем аренду DHCP

client = server.available();
if (client)
{
   while (client.connected())
   {
      if (client.available())
      {
         // Счетчики для количества запросов изменения состояния
         // контактов
         int digitalRequests = 0;
         int analogRequests = 0;
         if( client.find("GET /") ) // Ищем ключевое слово 'GET'
         {
            // Ищем метки, начинающиеся со слова "pin", и завершаем
            // поиск по достижении конца строки
            while(client.findUntil("pin", "\r\n"))
            {
               char type = client.read(); // Тип контакта D или A
               // Следующее целочисленное значение ASCII в тексте
               // представляет номер контакта
               int pin = client.parseInt();
               int val = client.parseInt(); // A целое число после
                                            // номера контакта будет его значением
               if( type == 'D')
               {
                  Serial.print("Digital pin "); // Цифровой контакт
                  pinMode(pin, OUTPUT);
                  digitalWrite(pin, val);
                  digitalRequests++;
               }
               else if( type == 'A')
               {
                  Serial.print("Analog pin "); // Аналоговый контакт
                  analogWrite(pin, val);
                  analogRequests++;
               }
               else
               {
                  Serial.print("Unexpected type "); // Неизвестный тип контакта
                  Serial.print(type);
               }
               Serial.print(pin);
               Serial.print("=");
               Serial.println(val);
            }
         }
         Serial.println();
```

```
            // Функция findUntil() обнаружила пустую строку (символ
            // новой строки (lf) с последующим символом
            // возврата каретки (cr))
            // Это означает конец запроса http, и теперь можно
            // отправлять ответ на него
            // Посылаем стандартный заголовок ответа HTTP
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println();

            // Отображаем в окне браузера количество контактов,
            // обработанных по запросу
            client.print(digitalRequests);
            client.print(" digital pin(s) written");
                        // Количество обработанных цифровых контактов
            client.println("<br />");
            client.print(analogRequests);
            client.print(" analog pin(s) written");
                          // Количество обработанных аналоговых контактов
            client.println("<br />");
            client.println("<br />");

            // Выводим входное значение каждого аналогового контакта
            for (int i = 0; i < 6; i++)
            {
               client.print("analog input ");
               client.print(i);
               client.print(" is ");
               client.print(analogRead(i));
               client.println("<br />");
            }
            break; // Принудительно завершаем цикл while()
         }
      }
      // Даем браузеру время, чтобы получить данные
      delay(100);
      client.stop();
   }
}
```

Обсуждение работы решения и возможных проблем

Исполняющийся скетч решения выводит в окно монитора порта URL-адрес, по которому можно отправлять запрос, с добавленными в конце параметрами запроса, — например: `http://IP_ADDRESS/?pinD2=1`. Здесь фрагмент *IP_ADDRESS* представляет действительный адрес шилда Ethernet, подключенного к плате Arduino, как отображено в окне монитора порта. Если ввести этот текст в адресную строку браузера и на-

жать клавишу <Enter>, то скетч установит на цифровом контакте 2 высокий логический уровень (значение 1). Запрос состоит из следующих частей:

- все, что находится перед символом знака вопроса, рассматривается как адрес веб-сервера — например: 192.168.1.177;
- за этой частью следует список полей данных, где каждое поле начинается с идентификатора `pin`, за которым следует символ `D` или `A`, обозначающий тип контакта: цифровой (Digital) или аналоговый (Analog) соответственно;
- далее следует номер контакта, затем знак равенства и собственно значение состояния, которое нужно присвоить этому контакту. Например, параметр `pinD2=1` означает, что цифровому контакту 2 нужно присвоить значение 1, т. е. установить на нем высокий логический уровень.

Запрос можно расширить, указав в нем несколько параметров, разделенных символом амперсанда (`&`). Например:

```
http://IP_ADDRESS/?pinD2=1&pinD3=0&pinA9=128&pinA11=255
```

Каждое поле между амперсандами обрабатывается таким же образом, как и первое поле. В запросе можно указывать любое количество полей, необходимых для изменения выходного состояния соответствующего количества контактов платы Arduino.

15.10. Обработка входящих запросов для конкретных страниц

ЗАДАЧА

Требуется организовать веб-сервер для раздачи нескольких разных страниц — например, для отображения показаний разных датчиков на разных страницах.

РЕШЕНИЕ

Эта задача решается скетчем, исходный код которого приводится в листинге 15.23. Скетч ищет в запросе ключевые слова `"analog"` и `"digital"` и выводит соответствующую страницу.

Так же, как и для скетчей из нескольких предыдущих решений, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующий заголовочный файл необходимо вписать правильное имя (SSID) и пароль используемой беспроводной сети.

Листинг 15.23. Скетч веб-сервера для раздачи нескольких разных веб-страниц

```
/*
* Скетч WebServerMultiPage
* Отвечает на отправляемые в URL-адресе запросы для просмотра состояния
  выходных цифровых и аналоговых контактов платы Arduino
```

```
* http://IP_ADDRESS/analog/ Веб-страница данных аналоговых контактов
* http://IP_ADDRESS/digital/ Веб-страница данных цифровых контактов
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для плат с шилдом Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

const int MAX_PAGE_NAME_LEN = 8;  // Максимальное количество символов
                                  // в названии страницы
char buffer[MAX_PAGE_NAME_LEN+1]; // Название страницы + конечный ноль

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

   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
         delay(0); // halt; ESP8266 does not like ∞ loop without a delay
                   // Останов. Платы на ESP8266 не любят бесконечных
                   // циклов без задержки
   }
   server.begin();
}

#define MSG_DELAY 10000
void loop()
{
   static unsigned long nextMsgTime = 0;
   if (millis() > nextMsgTime)
   {
      Serial.print("Try http://"); // Попробуйте ввести этот URL-адрес: http://
      Serial.print(getIP()); Serial.println("/analog/");
      nextMsgTime = millis() + MSG_DELAY;
   }
   maintain(); // При необходимости вручную обновляем аренду DHCP

   client = server.available();
   if (client)
   {
      while (client.connected())
      {
         if (client.available())
```

```
        {
          if( client.find("GET ") )
          {
            // Ищем название страницы
            memset(buffer,0, sizeof(buffer)); // Очищаем буфер
            if(client.find( "/"))
            if(client.readBytesUntil('/', buffer, MAX_PAGE_NAME_LEN ))
            {
              if(strcmp(buffer, "analog") == 0)
              showAnalog();
              else if(strcmp(buffer, "digital") == 0)
              showDigital();
              else
              unknownPage(buffer);
            }
          }
          Serial.println();
          break; // Принудительно завершаем цикл while()
        }
      }
      // Даем браузеру время, чтобы получить данные:
      delay(100);
      client.stop();
    }
}

void showAnalog()
{
  Serial.println("analog");
  sendHeader();
  client.println("<h1>Analog Pins</h1>");
  // Выводим входное значение каждого аналогового контакта
  for (int i = 0; i < 6; i++)
  {
    client.print("analog pin ");
    client.print(i);
    client.print(" = ");
    client.print(analogRead(i));
    client.println("<br />");
  }
}

void showDigital()
{
  Serial.println("digital");
  sendHeader();
  client.println("<h1>Digital Pins</h1>");
```

```
    // Выводим значение каждого цифрового контакта
    for (int i = 2; i < 8; i++)
    {
       pinMode(i, INPUT_PULLUP);
       client.print("digital pin ");
       client.print(i);
       client.print(" is ");
       if(digitalRead(i) == LOW)
          client.print("HIGH");
       else
          client.print("LOW");
       client.println("<br />");
    }
    client.println("</body></html>");
}

void unknownPage(char *page)
{
    sendHeader();
    client.println("<h1>Unknown Page</h1>");
    client.print(page);
    client.println("<br />");
    client.println("Recognized pages are:<br />");
    client.println("/analog/<br />");
    client.println("/digital/<br />");
    client.println("</body></html>");
}

void sendHeader()
{
    // Посылаем стандартный заголовок ответа HTTP
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println();
    client.println("<html><head><title>Web server multi-page Example</title>");
    client.println("<body>");
}
```

Обсуждение работы решения и возможных проблем

Чтобы протестировать работу этого скетча, введите в адресную строку браузера запрос: `http://`*`IP_ADDRESS`*`/analog/` или `http://IP_ADDRESS/digital/` (где идентификатор *`IP_ADDRESS`* представляет IP-адрес шилда Ethernet или платы ESP8266 или WiFiNINA, выведенный в окне монитора порта, — например: 192.168.1.127).

На рис. 15.3 показан пример ответа сервера. Для более глубокого тестирования можно подключить кнопки к одному или нескольким цифровым контактам и по-

тенциометры к одному или нескольким аналоговым контактам. Поскольку в скетче задействуются встроенные повышающие резисторы платы Arduino (см. *разд. 2.4*), его логика инвертирована. Иными словами, значение 0 для цифрового контакта будет означать, что кнопка нажата, а значение 1 — что не нажата. Подробная информация по работе с потенциометрами приводится в *разд. 5.6*.

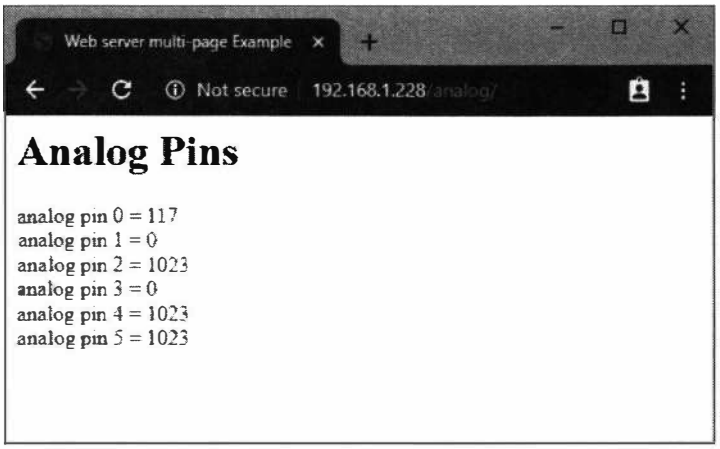


Рис. 15.3. Отображение в окне браузера ответа веб-сервера на запрос о состоянии аналоговых контактов платы Arduino

Скетч ищет в запросе символ косой черты (/), чтобы определить конец названия запрашиваемой страницы. Если название страницы не завершается косой чертой, сервер выдает сообщение о неизвестной странице (Unknown Page).

Код этого решения можно расширить, добавив в него фрагмент кода из скетча решения в *разд. 15.9* для управления выходными состояниями контактов платы Arduino из другой веб-страницы — с названием `update`. В листинге 15.24 приводится фрагмент кода основного цикла `loop()`, который нужно модифицировать (добавленные строки выделены полужирным шрифтом).

Листинг 15.24. Модификация скетча для установки значений контактов из другой веб-страницы

```
if(client.readBytesUntil('/', buffer, MAX_PAGE_NAME_LEN ))
{
   if(strcmp(buffer, "analog") == 0)
      showAnalog();
   else if(strcmp(buffer, "digital") == 0)
      showDigital();
   // Добавьте этот код для предоставления новой страницы с названием update
   else if(strcmp(buffer, "update") == 0)
      doUpdate();
   else
      unknownPage(buffer);
}
```

Исходный код функции `doUpdate()` приводится в листинге 15.25. Плата ESP8266 имеет ограниченное количество выходных контактов, узнать которые можно, изучив документацию на используемую плату. Подключение светодиодов к некоторым контактам может вызвать непредсказуемое поведение платы.

Листинг 15.25. Исходный код функции `doUpdate()`

```
void doUpdate()
{
   Serial.println("update");
   sendHeader();
   // Ищем метки, начинающиеся со слова "pin", и завершаем поиск
   // по обнаружению первой пустой строки
   while (client.findUntil("pin", "\n\r"))
   {
      char type = client.read(); // D or A
      int pin = client.parseInt();
      int val = client.parseInt();
      if ( type == 'D')
      {
         client.print("Digital pin ");
         pinMode(pin, OUTPUT);
         digitalWrite(pin, val);
      }
      else if ( type == 'A')
      {
         client.print("Analog pin ");
         analogWrite(pin, val);
      }
      else
      {
         client.print("Unexpected type "); // Неизвестный тип контакта
         Serial.print(type);
      }
      client.print(pin);
      client.print("=");
      client.println(val);
   }
   client.println("</body></html>");
}
```

Для установки состояния какого-либо контакта веб-серверу посылается запрос веб-страницы `update` с указанием в параметрах запроса контактов, для которых нужно задать состояние, и требуемое состояние. Например, запрос:

```
http://IP_ADDRESS/update/?pinA5=128
```

дает указание веб-серверу записать выходное значение 128 на аналоговый контакт 5.

Пакет поддержки плат на микроконтроллере ESP8266 содержит большой набор библиотек для создания веб-серверов. Если только вы не разрабатываете код для поддержки большого разнообразия сетевого оборудования, вам может оказаться полезным использовать эти возможности. Более подробная информация на эту тему приведена в *разд. 15.5*.

15.11. Использование HTML для форматирования ответов веб-сервера

ЗАДАЧА

Требуется использовать HTML-элементы — такие как таблицы и изображения, чтобы улучшить внешний вид веб-страниц, отправляемых веб-сервером на Arduino. Например, оформить результаты выполнения запроса решения из *разд. 15.10* в виде таблицы.

РЕШЕНИЕ

Эта задача решается с помощью скетча, исходный код которого приводится в листинге 15.26. На рис. 15.4 показан пример оформления результатов запроса этим скетчем. Сравните это оформление с неотформатированным представлением на рис. 15.3.

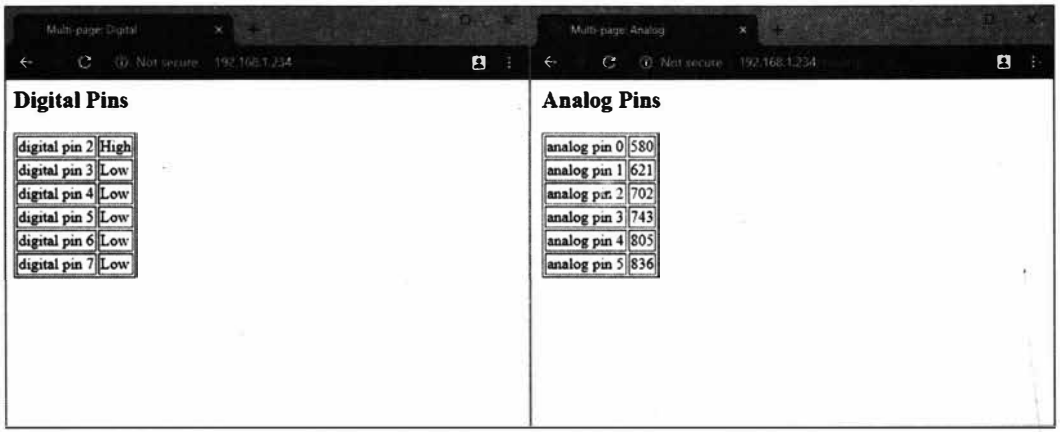


Рис. 15.4. Результаты запроса, отформатированные с помощью средств HTML

Точно так же, как и для скетча из предыдущего решения, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующем заголовочном файле необходимо указать правильное имя (SSID) и пароль используемой беспроводной сети. Плата ESP8266 имеет ограниченное количество выходных контактов, узнать которые можно, изучив документацию на используемую плату. Подключение светодиодов к некоторым контактам может вызвать непредсказуемое поведение платы.

Листинг 15.26. Скетч для форматирования результатов запроса с помощью средств HTML

```
/*
* Скетч WebServerMultiPageHTML
* Отображает в окне браузера значения аналоговых и цифровых контактов
  платы Arduino, отформатированные средствами HTML
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для шилдов Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

const int MAX_PAGE_NAME_LEN = 8;  // Максимальное количество символов
                                  // в названии страницы
char buffer[MAX_PAGE_NAME_LEN+1]; // Название страницы + конечный ноль

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

   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      delay(0); // halt; ESP8266 does not like ∞ loop without a delay
                // Останов. Платы на ESP8266 не любят бесконечных циклов без задержки
   }
   server.begin();
   pinMode(LED_BUILTIN, OUTPUT);
   for(int i=0; i < 3; i++)
   {
      digitalWrite(LED_BUILTIN, HIGH);
      delay(500);
      digitalWrite(LED_BUILTIN, LOW);
      delay(500);
   }
}

#define MSG_DELAY 10000
void loop()
{
   static unsigned long nextMsgTime = 0;
   if (millis() > nextMsgTime)
```

```
    {
        Serial.print("Try http://"); // Попробуйте ввести
                                    // этот URL-адрес: http://
        Serial.print(getIP()); Serial.println("/analog/");
        nextMsgTime = millis() + MSG_DELAY;
    }

    maintain(); // При необходимости вручную обновляем аренду DHCP

    client = server.available();
    if (client)
    {
        while (client.connected())
        {
            if (client.available())
            {
                if( client.find("GET ") )
                {
                    // Ищем название страницы
                    memset(buffer,0, sizeof(buffer)); // Очищаем буфер
                    if(client.find( "/"))
                    if(client.readBytesUntil('/', buffer, MAX_PAGE_NAME_LEN ))
                    {
                        if(strcasecmp(buffer, "analog") == 0)
                            showAnalog();
                        else if(strcasecmp(buffer, "digital") == 0)
                            showDigital();
                        else
                            unknownPage(buffer);
                    }
                }
                break;
            }
        }
        // Даем браузеру время, чтобы получить данные:
        delay(100);
        client.stop();
    }
}

void showAnalog()
{
    sendHeader("Multi-page: Analog");
    client.println("<h2>Analog Pins</h2>");
    client.println("<table border='1' >");
    for (int i = 0; i < 6; i++)
```

```
    {
      // Выводим значение каждого контакта аналогового ввода:
      client.print("<tr><td>analog pin ");
      client.print(i);
      client.print(" </td><td>");
      client.print(analogRead(i));
      client.println("</td></tr>");
    }
    client.println("</table>");
    client.println("</body></html>");
}

void showDigital()
{
    sendHeader("Multi-page: Digital");
    client.println("<h2>Digital Pins</h2>");
    client.println("<table border='1'>");
    for (int i = 2; i < 8; i++)
    {
      // Выводим значение каждого цифрового контакта
      pinMode(i, INPUT_PULLUP);
      digitalWrite(i, HIGH); // Включаем внутренние повышающие резисторы
      client.print("<tr><td>digital pin ");
      client.print(i);
      client.print(" </td><td>");
      if(digitalRead(i) == LOW)
        client.print("High");
      else
        client.print("Low");
      client.println("</td></tr>");
    }
    client.println("</table>");
    client.println("</body></html>");
}

void unknownPage(char *page)
{
    sendHeader("Unknown Page");  // Неизвестная страница
    client.println("<h1>Unknown Page</h1>");
    client.print(page);
    client.println("<br />");
    client.println("Recognized pages are:<br />");
                // Разрешаются следующие страницы:
    client.println("/analog/<br />");
    client.println("/digital/<br />");
    client.println("</body></html>");
}
```

```
void sendHeader(char *title)
{
   // Посылаем стандартный заголовок ответа HTTP:
   client.println("HTTP/1.1 200 OK");
   client.println("Content-Type: text/html");
   client.println();
   client.print("<html><head><title>");
   client.println(title);
   client.println("</title><body>");
}
```

Обсуждение работы решения и возможных проблем

Создаваемый скетчем веб-сервер предоставляет те же данные, что и в решении из *разд. 15.10*, но отформатированные в виде таблицы. Код для создания браузером таблицы с границей шириной в один пиксел выглядит так:

```
client.println("<table border='1' >");
```

Для созданной с его помощью таблицы цикл `for` посредством тега `<td>` определяет ячейки данных, а посредством тега `<tr>` — строки. Следующий код отображает строку `"analog pin"` в первой ячейке новой строки:

```
client.print("<tr><td>analog pin ");
```

А этот код отображает номер заданного контакта в этой же ячейке:

```
client.print(i);
```

Следующая строка кода содержит тег, который закрывает предыдущую ячейку и начинает новую:

```
client.print(" </td><td>");
```

Далее в этой новой ячейке отображается значение соответствующего аналогового контакта, возвращаемое функцией `analogRead()`:

```
client.print(analogRead(i));
```

После этого закрывается последняя ячейка строки и сама строка:

```
client.println("</td></tr>");
```

Цикл `for` повторяется до тех пор, пока не будут отображены значения всех шести аналоговых контактов.

Дополнительная информация

Дополнительную информацию по работе с HTML вы найдете в следующих книгах:

- «Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics», 5th edition («HTML, CSS, JavaScript и веб-графика. Подробное руко-

водство для начинающих», 5-е изд.), автор Jennifer Robbins (Дженнифер Роббинс), издательство O'Reilly[4];

- «Web Design in a Nutshell» («Краткое руководство по разработке веб-страниц»), автор Jennifer Robbins, издательство O'Reilly;
- «HTML & XHTML: The Definitive Guide» («HTML & XHTML. Подробное руководство»), авторы Chuck Musciano и Bill Kennedy, издательство O'Reilly.

15.12. Запрос данных посредством форм (метод POST)

ЗАДАЧА

Требуется создать веб-страницу с формами, предоставляющую пользователям возможность выбирать действие для выполнения платой Arduino. На рис. 15.5 показан пример созданной скетчем из этого решения такой веб-формы с кнопками.

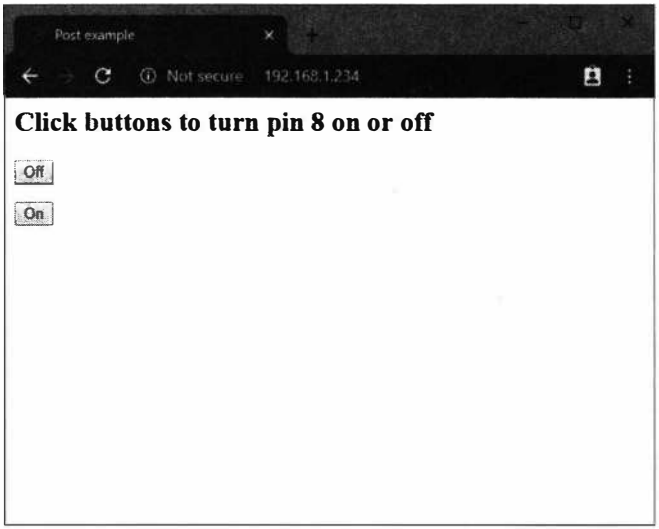


Рис. 15.5. Веб-форма с кнопками

РЕШЕНИЕ

Эта задача решается с помощью скетча, исходный код которого показан в листинге 15.27. Скетч создает веб-страницу, содержащую форму с двумя кнопками. При запросе этой веб-страницы в окне браузера отображаются две кнопки, нажатием на которые можно задавать соответствующие инструкции веб-серверу. В рассматриваемом случае скетч — в зависимости от нажатой кнопки — устанавливает высокий (`ON`) или низкий (`OFF`) уровень на контакте 8 платы Arduino.

---

[4] В издательстве «БХВ» ведется подготовка к выпуску русского издания этой книги. Следите за ее появлением в продаже на сайте издательства: **https://bhv.ru/**.

Листинг 15.27. Веб-форма для управления состоянием контакта платы Arduino

```
/*
* Скетч WebServerPost
* Включает или выключает контакт, используя HTML-форму
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для шилдов Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

const int MAX_PAGE_NAME_LEN = 8;  // Максимальное количество символов
                                  // в названии страницы
char buffer[MAX_PAGE_NAME_LEN+1]; // Название страницы + конечный ноль

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

   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      delay(0); // halt; ESP8266 does not like ∞ loop without a delay
                // Останов. Платы на ESP8266 не любят бесконечных циклов без задержки
   }
   server.begin();
}

#define MSG_DELAY 10000

void loop()
{
   static unsigned long nextMsgTime = 0;
   if (millis() > nextMsgTime)
   {
      Serial.print("Try http://"); // Попробуйте ввести
                                   // этот URL-адрес: http://
      Serial.println(getIP());
      nextMsgTime = millis() + MSG_DELAY;
   }

   maintain(); // При необходимости вручную обновляем аренду DHCP

   client = server.available();
```

```
if (client)
{
   int type = 0;
   while (client.connected())
   {
      if (client.available())
      {
         // GET, POST, or HEAD
         memset(buffer,0, sizeof(buffer)); // Очищаем буфер

         if(client.readBytesUntil('/', buffer,sizeof(buffer)))
         {
            Serial.println(buffer);
            if(strcmp(buffer,"POST ") == 0)
            {
               client.find("\r\n\r\n"); // Переходим к телу веб-страницы

               // Ищем метки, начинающиеся со слова "pin",
               // и завершаем поиск по обнаружению первой пустой строки
               // Параметры POST ожидаются в виде pinDx=y,
               // где x означает номер контакта,
               // а Y равно 0 для LOW или 1 для HIGH
               while(client.findUntil("pinD", "\r\n"))
               {
                  int pin = client.parseInt(); // Номер контакта
                  int val = client.parseInt(); // 0 или 1
                  pinMode(pin, OUTPUT);
                  digitalWrite(pin, val);
               }
            }
            else // Наверное, GET
            {
               if (client.find("favicon.ico")) // Отправляем ответ 404
                                               // для значка сайта
                  sendHeader("404 Not Found", "Not found");
            }
            sendHeader("200 OK", "Post example");

            // Создаем кнопку для выключения контакта 8
            client.println("<h2>Click buttons to turn pin 8 on or off</h2>");
            client.print(
               "<form action='/' method='POST'><p><input type='hidden' name='pinD8'");
            client.println(" value='0'><input type='submit' value='Off'/></form>");

            // Создаем кнопку для включения контакта 8
            client.print(
               "<form action='/' method='POST'><p><input type='hidden' name='pinD8'");
```

```
                client.print(" value='1'><input type='submit' value='On'/></form>");
                client.println("</body></html>");
                client.stop();
            }
            break; // Принудительно завершаем цикл while()
        }
    }
    // Даем браузеру время, чтобы получить данные:
    delay(100);
    client.stop();
  }
}

void sendHeader(char *code, char *title)
{
  // Посылаем стандартный заголовок ответа HTTP:
  client.print("HTTP/1.1 "); client.println(code);
  client.println("Content-Type: text/html");
  client.println();
  client.print("<html><head><title>");
  client.print(title);
  client.println("</title><body>");
}
```

Обсуждение работы решения и возможных проблем

Веб-страница с интерфейсом пользователя состоит из тегов HTML, которые определяют элементы управления (кнопки, метки, переключатели, поля для галочек и т. п.), составляющие этот интерфейс. В нашем решении для взаимодействия с пользователем используются кнопки.

Следующий фрагмент кода создает HTML-форму, содержащую кнопку с названием `pinD8`, обозначенную меткой `OFF`, при нажатии на которую серверу посылается значение `0`:

```
client.print(
    "<form action='/' method='POST'><p><input type='hidden' name='pinD8'");
client.println(" value='0'><input type='submit' value='Off'/></form>");
```

Получив запрос от клиента (браузера), сервер ищет в нем строку `"POST "`, чтобы определить начало переданной ему формы:

```
if (strcmp(buffer,"POST ") == 0) // Определяем начало переданной формы
  client.find("\r\n\r\n");       // Переходим к телу запроса
  // Ищем метки, начинающиеся со слова "pin",
  // и завершаем поиск по обнаружению первой пустой строки
  // Параметры POST ожидаются в виде pinDx=y,
  // где x означает номер контакта, а y равно 0 для LOW или 1 для HIGH
```

Полученная веб-сервером форма будет содержать строку `pinD8=0` при нажатой кнопке `OFF` или `pinD8=1` при нажатой кнопке `ON`.

Скетч выполняет поиск в теле запроса, пока не найдет название кнопки `pinD`:

```
while(client.findUntil("pinD", "\r\n"))
```

Метод (функция) `findUntil()` в этом коде ищет в переданном ему параметре подстроку `pinD` и прекращает поиск по нахождении этой подстроки или по достижении конца строки поиска (символов возврата каретки и новой строки `\r\n`, вставленных браузером в конце формы).

Число, следующее за подстрокой `pinD`, является номером контакта:

```
int pin = client.parseInt();  // Номер контакта
```

А следующее за ним значение будет `0` — при нажатии кнопки `OFF`, или `1` — при нажатии кнопки `ON`:

```
int val = client.parseInt();      // 0 или 1
```

Для контакта задается режим записи, а затем в него записывается полученное значение:

```
pinMode(pin, OUTPUT);
digitalWrite(pin, val);
```

В форме можно создавать дополнительные кнопки, используя соответствующие теги. Например, следующий фрагмент кода добавляет кнопку для включения цифрового контакта 9:

```
// Создаем кнопку для включения контакта 9
client.print(
    "<form action='/' method='POST'><p><input type='hidden' name='pinD9'");
client.print(" value='1'><input type='submit' value='On'/></form>");
```

15.13. Раздача веб-страниц, содержащих большие объемы данных

ЗАДАЧА

Созданная вами веб-страница требует больше памяти, чем доступный для этого объем памяти SRAM, поэтому для ее хранения нужно использовать флеш-память, также называемую *программной памятью* (см. решение из *разд. 17.4*).

РЕШЕНИЕ

Эта задача решается с помощью скетча, исходный код которого приводится в листинге 15.28. Скетч совмещает в себе код передачи запросов методом POST из решения *разд. 15.12* с кодом, использующим теги HTML для форматирования ответа из решения *разд. 15.11*, а также содержит новый код для доступа к данным, хранящимся во флеш-памяти. Так же, как и в решении из *разд. 15.11*, в этом решении

веб-сервер отображает состояние цифровых и аналоговых контактов и включает и выключает цифровые контакты. Пример создаваемой этим решением страницы для управления состоянием контактов и отображения их состояния приведен на рис. 15.6.

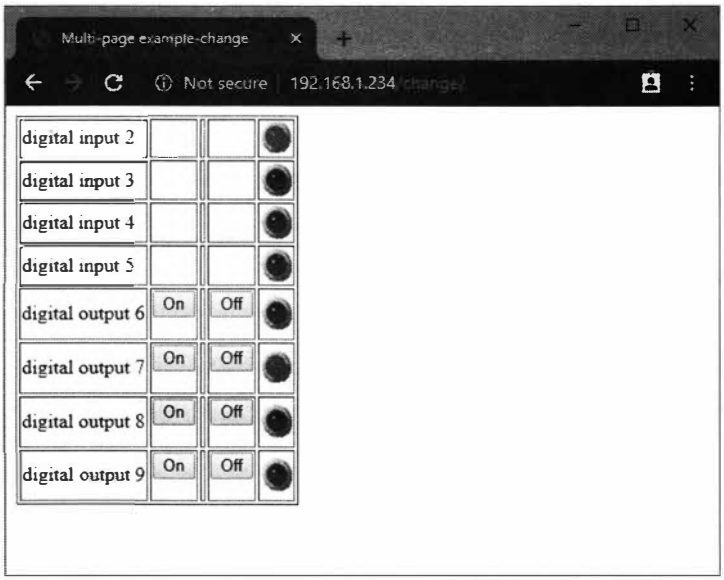


Рис. 15.6. Веб-страница для управления состоянием контактов платы Arduino и отображения этого состояния

Точно так же, как и для скетча из предыдущих решений, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующем заголовочном файле необходимо указать правильное имя (SSID) и пароль используемой беспроводной сети. Плата ESP8266 имеет ограниченное количество выходных контактов, узнать которые можно, изучив документацию на используемую плату. Подключение светодиодов к некоторым контактам может вызвать непредсказуемое поведение платы.

Листинг 15.28. Скетч для управления состоянием контактов платы Arduino и отображения этого состояния

```
/*
* Скетч WebServerMultiPageHTMLProgmem
*

* Отвечает на отправляемые в URL-адресе запросы на изменение состояния
  выходных цифровых и аналоговых контактов платы Arduino
* Отображает в браузере количество портов с измененными состояниями
  и входные значения аналоговых портов
*
```

```
* http://192.168.1.177/analog/   Веб-страница для отображения данных
                                 аналоговых контактов
* http://192.168.1.177/digital/  Веб-страница для отображения данных
                                 цифровых контактов
* http://192.168.1.177/change/   Веб-страница для изменения данных
                                 цифровых контактов
*
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для шилдов Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

#include <avr/pgmspace.h>    // Для работы с программной памятью
#define P(name) static const char name[] PROGMEM // Объявляется статическая строка
const int MAX_PAGENAME_LEN = 8;  // Максимальное количество символов
                                 // в названии страницы
char buffer[MAX_PAGENAME_LEN+1]; // Дополнительный символ
                                 // для завершающего нуль-символа

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

   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
         delay(0); // Останов. Платы на ESP8266 не любят бесконечных
                   // циклов без задержки
   }
   server.begin();
   Serial.println(F("Ready"));
}

#define MSG_DELAY 10000

void loop()
{
   static unsigned long nextMsgTime = 0;
   if (millis() > nextMsgTime)
   {
      Serial.print("Try http://"); // Попробуйте ввести
                                   // этот URL-адрес: http://
      Serial.print(getIP()); Serial.println("/change/");
```

```
      nextMsgTime = millis() + MSG_DELAY;
   }

   maintain(); // При необходимости вручную обновляем аренду DHCP

   client = server.available();
   if (client)
   {
      int type = 0;
      while (client.connected())
      {
         if (client.available())
         {
            // GET, POST, or HEAD
            memset(buffer,0, sizeof(buffer)); // Очищаем буфер
            if(client.readBytesUntil('/', buffer,MAX_PAGENAME_LEN))
            {
               if(strcmp(buffer, "GET ") == 0 )
                  type = 1;
               else if(strcmp(buffer,"POST ") == 0)
                  type = 2;
               // Ищем название страницы
               memset(buffer,0, sizeof(buffer)); // clear the buffer
               if(client.readBytesUntil( '/', buffer,MAX_PAGENAME_LEN ))
               {
                  if(strcasecmp(buffer, "analog") == 0)
                     showAnalog();
                  else if(strcasecmp(buffer, "digital") == 0)
                     showDigital();
                  else if(strcmp(buffer, "change")== 0)
                     showChange(type == 2);
                  else
                     unknownPage(buffer);
               }
            }
            break;
         }
      }
      // Даем браузеру время, чтобы получить данные:
      delay(100);
      client.stop();
   }
}

void showAnalog()
{
   Serial.println(F("analog"));
```

```
   sendHeader("Multi-page example-Analog");
   client.println("<h1>Analog Pins</h1>");
   // Выводим значение каждого контакта аналогового ввода
   client.println(F("<table border='1' >"));

   for (int i = 0; i < 6; i++)
   {
      client.print(F("<tr><td>analog pin "));
      client.print(i);
      client.print(F(" </td><td>"));
      client.print(analogRead(i));
      client.println(F("</td></tr>"));
   }
   client.println(F("</table>"));
   client.println(F("</body></html>"));
}

// Данные в кодировке MIME для изображений
// включенного и выключенного светодиода
// См. http://www.motobit.com/util/base64-decoder-encoder.asp
P(led_on) = "<img src=\""
```

```
"/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAHgAA/+4ADkFkb2JlAGTAAAAAAf/b"
"AIQAEAsLCwwLEAwMEBcPDQ8XGxQQEBQbHxcXFxcXHx4XGhoaGhceHiMlJyUjHi8vMzMvL0BAQEBA"
"QEBAQEBAQEBAQAERDw8RExEVEhIVFBEUERQaFBYWFBomGhocGhomMCMeHh4eIzArLicnJy4rNTUw"
"MDU1QEA/QEBAQEBAQEBAQEBA/8AAEQgAGwAZAwEiAAIRAQMRAf/EAIIAAAICAwAAAAAAAAAAAAAA"
"AAUGAAcCAwQBAAMBAAAAAAAAAAAAAAAAAACBAUQAAECBAQBCgcAAAAAAAAAAAAECAwARMRIhQQQF"
"UWFxkaHRMoITUwYiQnKSIxQ1EQAAAwYEBwAAAAAAAAAAAAAARECEgMTBBQhQWEiMVGBMkJiJP/a"
"AAwDAQACEQMRAD8AcNz3BGibKie0nhC0v3A+teKJt8JmZEdHuZal0itgUoHnEpQEWtSyLqgACWFI"
"nixWiaQhsUFFBiQSbiMvvrmeCBp27eLnG7lFTDxs+Kra8oOyium3ltJUAcDIy4EUMN/7Dnq9cPMO"
"W90E9kxeyF2d3HFOQ175olKudUm7TqlfKqDQEDOFR1sNqtC7k5ERYjndNPFSArtvnI/nV+ed9coI"
"ktd2BgozrSZO3J5jVEXRcwD2bbXNdq0zT+BohTyjgPp5SYdPJZ9NP2jsiIz7vhjLohtjnqJ/ouPK"
"co//2Q=="
```

```
"\"/>";

P(led_off) = "<img src=\""
```

```
"/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAHgAA/+4ADkFkb2JlAGTAAAAAAf/b"
"AIQAEAsLCwwLEAwMEBcPDQ8XGxQQEBQbHxcXFxcXHx4XGhoaGhceHiMlJyUjHi8vMzMvL0BAQEBA"
"QEBAQEBAQEBAQAERDw8RExEVEhIVFBEUERQaFBYWFBomGhocGhomMCMeHh4eIzArLicnJy4rNTUw"
"MDU1QEA/QEBAQEBAQEBAQEBA/8AAEQgAHAAZAwEiAAIRAQMRAf/EAHgAAQEAAwAAAAAAAAAAAAAA"
"AAYFAgQHAQEBAQAAAAAAAAAAAAAAAAAAACAQQQAAECBQAHBQkAAAAAAAAAAAECAwAREhMEITFhoSIF"
"FUFR0UIGgZHBMlIjM1MWEQABAwQDAQEAAAAAAAAAAAABABECIWESA1ETIyIE/9oADAMBAAIRAxEA"
"PwBvl5SWEkkylpJMGsj1XjXSE1kCQuJ8Iy9W5DoxradFa6VDf8IJZAQ6loNtBooTJaqp3DP5oBlV"
"nWrTpEouQS/Cf4PO0uKbqWHGXTSlztSvuVFiZjmfLH3GUuMkzSoTMu8aiNsXet5/17hFyo6PR64V"
"ZnuqfqDDDySFpNpYH3E6aFjzGBr2DkMuFBSFDsWkilUdLftW13pWpcdWqnbBzI/l6hVXKZlROUSe"
"L1KX5zvAPXESjdHsTFWpxLKOJ54hIA1DZCj+Vx/3r96fCNrkvRaT0+V3zV/llplr9sVeHZui/ONk"
"H3dzt6cL/9k="
```

```
"\"/>";
```

```
void showDigital()
{
   Serial.println(F("digital"));
   sendHeader("Multi-page example-Digital");
   client.println(F("<h2>Digital Pins</h2>"));
   // show the value of digital pins
   client.println(F("<table border='1'>"));
   for (int i = 2; i < 8; i++)
   {
      pinMode(i, INPUT_PULLUP);
      client.print(F("<tr><td>digital pin "));
      client.print(i);
      client.print(F(" </td><td>"));
      if(digitalRead(i) == HIGH)
         printP(led_off);
      else
         printP(led_on);
      client.println(F("</td></tr>"));
   }
   client.println(F("</table>"));
   client.println(F("</body></html>"));
}

void showChange(bool isPost)
{
   Serial.println(F("change"));
   if(isPost)
   {
      Serial.println("isPost");
      client.find("\r\n\r\n"); // Перейти к телу запроса
      // Ищем метки, начинающиеся со слова "pin",
      // и завершаем поиск по обнаружению первой пустой строки
      Serial.println(F("searching for parms"));
      while(client.findUntil("pinD", "\r\n"))
      {
         int pin = client.parseInt(); // Номер контакта
         int val = client.parseInt(); // 0 или 1
         Serial.print(pin);
         Serial.print("=");
         Serial.println(val);
         pinMode(pin, OUTPUT);
         digitalWrite(pin, val);
      }
   }
   sendHeader("Multi-page example-change");

   // Таблица с кнопками для контактов со 2-го по 9-й
   // Контакты со 2-го по 5-й — вводы, остальные — выводы
   client.println(F("<table border='1'>"));
```

```
    // Отображаем контакты ввода
    for (int i = 2; i < 6; i++) // Контакты 2-5 – вводы
    {
       pinMode(i, INPUT_PULLUP);
       client.print(F("<tr><td>digital input "));
       client.print(i);
       client.print(F(" </td><td>"));

       client.print(F("  </td><td>"));
       client.print(F(" </td><td>"));
       client.print(F("  </td><td>"));

       if(digitalRead(i) == HIGH)
          printP(led_off);
       else
          printP(led_on);
       client.println("</td></tr>");
    }

    // Отображаем контакты вывода 6-9
    // Обратите внимание, что контакты 10-13 используются шилдом Ethernet
    for (int i = 6; i < 10; i++)
    {
       client.print(F("<tr><td>digital output "));
       client.print(i);
       client.print(F(" </td><td>"));
       htmlButton( "On", "pinD", i, "1");
       client.print(F(" </td><td>"));
       client.print(F(" </td><td>"));
       htmlButton("Off", "pinD", i, "0");
       client.print(F(" </td><td>"));

       if(digitalRead(i) == LOW)
          printP(led_off);
       else
          printP(led_on);
       client.println(F("</td></tr>"));
    }
    client.println(F("</table>"));
}

// Создаем HTML кнопку
void htmlButton( char * label, char *name, int nameId, char *value)
{
    client.print(F(
          "<form action='/change/' method='POST'><p><input type='hidden' name='"));
    client.print(name);
    client.print(nameId);
```

```
    client.print(F("' value='"));
    client.print(value);
    client.print(F("'><input type='submit' value='"));
    client.print(label);
    client.print(F("'/></form>"));
}

void unknownPage(char *page)
{
    Serial.print(F("Unknown : "));
    Serial.println(F("page"));

    sendHeader("Unknown Page");
    client.println(F("<h1>Unknown Page</h1>"));
    client.println(page);
    client.println(F("</body></html>"));
}

void sendHeader(char *title)
{
    // Посылаем стандартный заголовок ответа HTTP:
    client.println(F("HTTP/1.1 200 OK"));
    client.println(F("Content-Type: text/html"));
    client.println();
    client.print(F("<html><head><title>"));
    client.println(title);
    client.println(F("</title><body>"));
}

void printP(const char *str)
{
    // Копируем данные из программной памяти в локальное хранилище
    // Записываем фрагментами по 32 байта, чтобы избежать
    // слишком коротких пакетов TCP/IP
    // Взято из библиотеки webduino.
    // Copyright 2009 Ben Combee, Ran Talbott

    uint8_t buffer[32];
    size_t bufferEnd = 0;

    while (buffer[bufferEnd++] = pgm_read_byte(str++))
    {
        if (bufferEnd == 32)
        {
            client.write(buffer, 32);
            bufferEnd = 0;
        }
    }
```

```
    // Записываем все, что осталось, за исключением завершающего
    // нуль-символа
    if (bufferEnd > 1)
      client.write(buffer, bufferEnd - 1);
}
```

Обсуждение работы решения и возможных проблем

Логика создания веб-страницы в этом решении подобна логике из предыдущих решений. Используемая здесь форма основана на коде формы решения из *разд. 15.12*, но она содержит большее количество элементов в таблице, а также использует встроенные графические элементы для представления состояния контактов.

Если вам когда-либо приходилось создавать веб-страницы, то вы, возможно, знакомы с включением в состав страницы JPEG-изображений. Однако библиотеки Ethernet не поддерживают возможность работы с изображениями в формате *.jpg. Поэтому изображения нужно закодировать, используя один из интернет-стандартов, — например, MIME-кодирование. Это кодирование позволяет представлять графические и другие материалы с помощью текстовых строк. В скетче этого решения можно видеть, как выглядят изображения светодиодов, закодированные в MIME-кодировке. Многие веб-сайты предоставляют услугу кодирования изображений в MIME-формат. Для кодирования изображений в этом решении мы воспользовались услугами службы: **https://oreil.ly/7YWLw**.

Но размер описаний даже таких небольших изображений, как светодиоды, которые присутствуют в этом примере, слишком большой, чтобы поместиться в памяти данных SRAM микроконтроллера AVR. Поэтому для их хранения задействуется программная (флеш) память (выражение `P(name)` подробно рассматривается в *разд. 17.3*). Эта возможность используется только в тех случаях, когда скетч исполняется на платах с микроконтроллером AVR. Платы Arduino с 32-разрядными микроконтроллерами могут сохранять в памяти данных больший объем данных, а компилятор Arduino достаточно смышленый, чтобы решить, где хранить статические строковые данные.

Изображения включенного и выключенного светодиода сохраняются в виде последовательности символов. Массив для хранения изображения включенного светодиода объявляется следующим образом:

```
P(led_on) = "<img src=\""
```

Оператор `P(led_on)` присваивает этому массиву название `led_on`. Следующий за знаком равенства тег HTML идентифицирует изображение, а за ним следуют данные в MIME-кодировке, определяющие изображение.

Скетч этого решения основан на коде для веб-сервера Webduino. Хотя на момент подготовки материала книги поддержка этого сервера была прекращена, он может оказаться полезным для создания веб-страниц более сложных, чем показанные в этой главе примеры.

Дополнительная информация

Обратитесь к *разд. 17.4* за дополнительной информацией по использованию конструкции `F("text")` для сохранения текста во флеш-памяти.

Информацию по веб-серверу Webduino вы найдете на веб-сайте GitHub: **https://oreil.ly/LJodK**.

15.14. Отправка сообщений в Twitter

ЗАДАЧА

Требуется организовать отправку сообщений в Twitter средствами платформы Arduino. Например, для мониторинга показаний какого-либо датчика.

РЕШЕНИЕ

Эта задача решается с помощью скетча из листинга 15.29. Скетч отправляет сообщение в Twitter при замыкании контактов переключателя. Для авторизации на сайте Twitter он использует службу прокси ThingSpeak (**http://www.thingspeak.com**), поэтому вам нужно предварительно зарегистрироваться на этом сайте, чтобы получить бесплатный ключ API-интерфейса. Для этого просто нажмите кнопку **Get Started for Fee** на домашней странице сайта и заполните форму. Создав учетную запись, вы получите ключ API-интерфейса службы ThingSpeak. Кроме того, чтобы использовать службу ThingSpeak, необходимо дать указание вашей учетной записи Twitter, чтобы службе ThingTweet было разрешено добавлять сообщения в вашу учетную запись (это можно сделать на странице ThingsTweets (**https://oreil.ly/5UZrP**) после создания учетной записи ThingSpeak). Выполнив все эти предварительные требования и получив свой ключ API-интерфейса службы, замените в скетче текст `"YourThingTweetAPIKey"` на этот ключ.

Точно так же, как и для скетчей из нескольких предыдущих решений, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующем заголовочном файле необходимо указать правильное имя (SSID) и пароль используемой беспроводной сети.

Листинг 15.29. Скетч для создания записей в Twitter

```
/*
* Скетч ThingTweet
* Делает запись в Twitter при нажатии кнопки, подключенной к контакту 2
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"       // Для плат WiFiNINA
```

```
//#include "USE_Ethernet.h"    // Для шилдов Ethernet
//#include "USE_ESP8266.h"     // Для плат ESP8266

// Ваш ключ API-интерфейса ThingTweet
char *thingtweetAPIKey = "YourThingTweetAPIKey";
char serverName[] = "api.thingspeak.com";

bool MsgSent = false;
const int btnPin = 2;

void setup()
{
   Serial.begin(9600);
   while (!Serial);
   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while (1)
      delay(0); // Останов. Платы на ESP8266 не любят бесконечных
               // циклов без задержки
   }
   pinMode(btnPin, INPUT_PULLUP);
   delay(1000);
   Serial.println("Ready");
}

void loop()
{
   if (digitalRead(btnPin) == LOW) // Переходим сюда, если нажата кнопка
   {
      if (MsgSent == false) // Проверяем, не отправлено ли уже сообщение
      {
         MsgSent = sendMessage("I pressed a button on something #thingspeak");
                        // Я нажал кнопку на чем-то #thingspeak"
         if (MsgSent)
            Serial.println("Tweeted successfully"); // Запись
                                                    // успешно добавлена
         else
            Serial.println("Unable to Tweet"); // Не удалось добавить запись
      }
   }
   else
   {
      MsgSent = false; // Кнопка не нажата
   }
   delay(100);
}
```

```
bool sendMessage(char *message)
{
   bool result = false;
   const int tagLen = 16;   // Количество символов тегов, использующихся
                            // для обрамления сообщения
   int msgLen = strlen(message) + tagLen + strlen(thingtweetAPIKey);
   Serial.println("Connecting ..."); // Подключаемся...
   if (client.connect(serverName, 80) )
   {
      Serial.println("Making POST request..."); // Посылаем запрос POST
      client.println("POST /apps/thingtweet/1/statuses/update HTTP/1.1");
      client.print("Host: "); client.println(serverName);
      client.println("Connection: close");
      client.println("Content-Type: application/x-www-form-urlencoded");
      client.print("Content-Length: "); client.println(msgLen);
      client.println();
      client.print("api_key="); // Тег сообщения
      client.print(thingtweetAPIKey); // Ключ API-интерфейса
      client.print("&status=");       // Тег сообщения
      client.print(message);          // Сообщение
      client.println();
   }
   else
   {
      Serial.println("Connection failed"); // Ошибка подключения
   }
   // response string
   if (client.connected())
   {
      Serial.println("Connected"); // Подключились
      if (client.find("HTTP/1.1") && client.find("200 OK") )
      {
         result = true;
      }
      else
      Serial.println("Dropping connection - no 200 OK");
                  // Разрываем подключение — нет сообщения 200 OK
   }
   else
   {
      Serial.println("Disconnected"); // Отключились
   }
   client.stop();
   client.flush();
   return result;
}
```

Обсуждение работы решения и возможных проблем

Скетч отслеживает состояние контакта 2 и при обнаружении на нем низкого уровня (`LOW`) создает запись в вашей учетной записи Twitter, используя для этого API-интерфейс веб-службы ThingTweet.

Взаимодействие с веб-интерфейсом обрабатывается функцией `sendMessage()`, которая передает в Twitter указанную строку сообщения. В нашем скетче эта функция пытается передать строку `"Mail has been delivered"` (Почта доставлена) и возвращает булево значение `true` при успешном подключении.

Дополнительную информацию по службе ThingTweet ищите в ее документации (**https://oreil.ly/QNT5h**).

Дополнительная информация

Дополнительная информация по службе ThingSpeak приводится также в учебном руководстве для Arduino (**https://oreil.ly/ev78V**).

Информация по библиотеке Twitter Arduino, которая позволяет прямое взаимодействие с Twitter, приведена на веб-странице справки для этой библиотеки (**https://oreil.ly/dNHMr**).

15.15. Организация обмена данными для Интернета вещей

ЗАДАЧА

Требуется организовать обмен данными между устройствами, соединенными через Интернет.

РЕШЕНИЕ

Эту задачу можно решить, используя для реализации обмена данными платами Arduino, подключенными к Интернету, протокол MQTT (Message Queue Telemetry Transport — передача последовательности сообщений с телеметрическими данными). Упрощенный быстродействующий протокол MQTT предназначен для реализации отправки (издания) и получения (подписки) данных. Им легко пользоваться, и он хорошо исполняется на платформе Arduino, что делает его вполне подходящим для использования в проектах Интернета вещей.

Для передачи опубликованных данных клиентам протокол MQTT использует подключенный к Интернету сервер, который называется *брокером*. Создатели данных отправляют брокеру сообщения на определенную тему (публикуют их). Потребители данных подключаются к брокеру и подписываются на одну или несколько тем. Брокер сортирует полученные сообщения по темам и направляет сообщения на каждую тему всем потребителям, которые подписались на ту или иную тему.

Хотя платы с ограниченным объемом памяти (например, Arduino Uno) могут как публиковать данные, так и подписываться на сообщения, их вычислительные ресурсы слишком ограничены для исполнения приложения брокера. Приложение брокера можно исполнять на компьютере под Windows, Linux или macOS, или же использовать какой-либо из бесплатных общедоступных «облачных» брокеров — например, **mqtt.eclipse.org** и **test.mosquitto.org**. Более обширный список брокеров MQTT предлагается здесь: **https://oreil.ly/vnniF**.

В скетчах следующих далее решений демонстрируется подключение к брокеру Eclipse. Информацию о подключении к другим общедоступным брокерам MQTT можно найти на их веб-сайтах.

Для тех, кто хочет установить брокер на свой компьютер, можно порекомендовать популярный брокер с открытым исходным кодом Mosquitto (**https://mosquitto.org**).

Обсуждение работы решения и возможных проблем

Для взаимодействия с брокером MQTT скетч Arduino ориентируется на одну из предназначенных для этой цели библиотек. Можно порекомендовать следующие две популярные библиотеки брокера MQTT, устанавливаемые с помощью Менеджера библиотек:

- PubSubClient (**https://oreil.ly/_lW-B**), разработчик Nick O'Leary;
- Библиотека Adafruit MQTT (**https://oreil.ly/Z3GKn**).

В следующих двух разделах показано, как публиковать сообщения и подписываться на них, используя библиотеку PubSubClient.

Дополнительная информация

Дополнительную информацию по протоколу MQTT вы найдете на его веб-сайте (**http://mqtt.org**).

15.16. Публикация данных на брокере MQTT

ЗАДАЧА

Требуется организовать публикацию данных на брокере MQTT.

РЕШЕНИЕ

Эта задача решается с помощью скетча, исходный код которого приводится в листинге 15.30. Скетч использует библиотеку PubSubClient (разработчик Nick O'Leary) и публикует выходное значение аналогового контакта 0 в теме `esp/alog`. Библиотека устанавливается с помощью Менеджера библиотек среды Arduino IDE (подробно об установке библиотек рассказано в *главе 16*).

Точно так же, как и для скетчей из нескольких предыдущих решений, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать

в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующем заголовочном файле необходимо указать правильное имя (SSID) и пароль используемой беспроводной сети.

Листинг 15.30. Скетч для публикации данных на брокере MQTT

```
/*
* Скетч Basic MQTT publish
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"      // Для плат WiFiNINA
//#include "USE_Ethernet.h"  // Для шилдов Ethernet
//#include "USE_ESP8266.h"   // Для плат ESP8266

#include <PubSubClient.h>

const char* broker = "mqtt.eclipse.org"; // Адрес брокера MQTT

const int interval = 5000;   // Интервалы между событиями в миллисекундах
unsigned int timePublished;  // Время по функции millis() самой последней публикации

PubSubClient psclient(client);

void setup()
{
   Serial.begin(9600);
   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
      while(1)
      delay(0); // Останов. Платы на ESP8266 не любят бесконечных циклов без задержки
   }
   psclient.setServer(broker, 1883);
}

void loop(void)
{
   if (millis() - timePublished > interval)
   {
      if (!psclient.connected())
      psclient.connect("arduinoCookbook3Pub");
      if (psclient.connected())
      {
         int val = analogRead(A0);
         psclient.publish("arck3/alog", String(val).c_str());
         timePublished = millis();
```

```
            Serial.print("Published "); Serial.println(val);
        }
    }
    if (psclient.connected())
    psclient.loop();
}
```

Обсуждение работы решения и возможных проблем

Переменная `broker` служит для хранения адреса брокера MQTT, к которому требуется подключаться. В приведенном решении ей присваивается значение `mqtt.eclipse.org`. Адрес и порт брокера устанавливаются следующим кодом:

```
const char* broker = "mqtt.eclipse.org"; // Адрес брокера MQTT
psclient.setServer(broker, 1883);
```

По умолчанию для подключения к брокеру MQTT используется порт 1883. Но на всякий случай уточните в документации на свой брокер, не нужно ли для него использовать какой-либо другой порт.

Код в главном цикле `loop()` проверяет, прошло ли достаточно времени для публикации другого образца, и если да, переменной `val` присваивается значение аналогового контакта 0. Затем вызывается функция для публикации данных, которой передается строка названия темы и публикуемое значение:

```
psclient.publish("arck3/alog", String(val).c_str());
```

Метод (функция) ожидает С-строку (т. е. последовательность символов, завершающуюся символом `NULL`), выражение `String(val).c_str()` преобразовывает целочисленное значение, полученное в результате считывания аналогового сигнала, в данные типа `String` Arduino и возвращает их в виде С-строки.

Для просмотра опубликованных данных требуется MQTT-клиент с подпиской на тему `esp/alog`. Один из возможных клиентов рассматривается в следующем разделе.

Дополнительная информация

Более подробная информация по типу данных `String` Arduino приводится в *разд. 2.5*.

15.17. Подписка на данные брокера MQTT

ЗАДАЧА

Требуется организовать подписку на получение данных с брокера MQTT.

РЕШЕНИЕ

Эта задача решается с помощью скетча из листинга 15.31. Так же, как и в скетче из предыдущего решения, в этом скетче используется библиотека PubSubClient для создания подписки на данные, публикуемые предыдущим решением.

Точно так же, как и для скетчей из нескольких предыдущих решений, для этого скетча требуется создать один из трех заголовочных файлов и раскомментировать в скетче строку `#define`, соответствующую используемому сетевому интерфейсу. Для плат на микроконтроллере ECP8266 или плат WiFiNINA в соответствующем заголовочном файле необходимо указать правильное имя (SSID) и пароль используемой беспроводной сети.

Листинг 15.31. Скетч для создания подписки на брокере MQTT

```
/*
* Скетч Basic MQTT subscribe
*/

// Раскомментируйте только одну из следующих трех строк кода
//#include "USE_NINA.h"       // Для плат WiFiNINA
//#include "USE_Ethernet.h"   // Для шилдов Ethernet
//#include "USE_ESP8266.h"    // Для плат ESP8266

#include <PubSubClient.h>

const char* broker = "mqtt.eclipse.org"; // Адрес брокера MQTT
const int interval = 5000;  // Интервалы между событиями в миллисекундах
unsigned int timePublished; // Время самой последней публикации

PubSubClient psclient(client);

void callback(char* topic, byte* payload, unsigned int length)
{
   Serial.print("Message on topic [");
   Serial.print(topic);
   Serial.print("] ");
   for (int i=0; i < length; i++)
   {
      Serial.write(payload[i]);
   }
   Serial.println();
}

void setup()
{
   Serial.begin(9600);
   if (!configureNetwork()) // Инициализируем сеть
   {
      Serial.println("Failed to configure the network");
                  // Не удалось выполнить настройку сети
```

```
      while(1)
         delay(0); // Останов. Платы на ESP8266 не любят бесконечных
                   // циклов без задержки
   }
   psclient.setServer(broker, 1883);
   psclient.setCallback(callback);
}

void loop(void)
{
   if (!psclient.connected())
   {
      if (psclient.connect("arduinoCookbook3Sub"))
      {
         Serial.println("Subscribing to arck3/alog");
         psclient.subscribe("arck3/alog");
      }
   }
   if (psclient.connected())
   psclient.loop();
}
```

Подключение к брокеру в этом решении осуществляется подобно тому, как это делалось в скетче предыдущего решения. Отличия между остальными функциональностями следующие:

- в этом решении определяется функция обратного вызова `callback()`, которая вызывается при публикации события на тему подписки;
- этот скетч посылает другой идентификатор (`arduinoCookbook3Sub`) при вызове функции `connect()`, чтобы позволить однозначно идентифицировать его;
- в главном цикле `loop()` этого решения реализуется *подписка* (получение данных) на определенную тему. В главном цикле `loop()` скетча предыдущего решения реализуется *публикация* (отправка) данных на определенную тему.

Обсуждение работы решения и возможных проблем

Функция обратного вызова представляет собой функцию, которая передается другой функции и исполняется, когда вторая функция принимает решение об ее исполнении. В нашем случае такой другой функцией является объект `psclient` библиотеки `PubSubClient`, который исполняет функцию обратного вызова `callback()` при публикации данных на тему подписки.

Функции обратного вызова в параметрах передается тема и полезная нагрузка. Темой является строка, конец которой определяется завершающим нуль-символом. Но данные полезной нагрузки также могут содержать нуль-символы, поэтому для индикации конца сообщения предоставляется значение количества символов данных.

15.18. Получение значения текущего времени от интернет-сервера времени

ЗАДАЧА

Требуется получать текущее время от интернет-сервера времени — например, чтобы синхронизировать приложение часов, исполняющееся на плате Arduino.

РЕШЕНИЕ

Эта задача решается с помощью скетча, исходный код которого приводится в листинге 15.32. Скетч получает текущее время от сервера NTP (Network Time Protocol — сетевой протокол времени) и выводит полученное значение в окне монитора порта в виде количества секунд, истекших с 1 января 1900 года (время NTP) и с 1 января 1970 года.

Листинг 15.32. Скетч для получения текущего времени от интернет-сервера времени

```
/*
* Скетч UdpNtp
* Получает значения текущего времени от интернет-сервера времени
* Демонстрирует использование отправки и получения пакетов по протоколу UDP
*/

#include <SPI.h>
#include <Ethernet.h>

#include <EthernetUDP.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // Уникальный MAC-адрес

unsigned int localPort = 8888; // Локальный порт, на котором ожидаются UDP-сообщения

IPAddress timeServer(192, 43, 244, 18); // Сервер NTP time.nist.gov
const int NTP_PACKET_SIZE= 48;          // Временной меткой NTP являются
                                        // первые 48 байтов сообщения
byte packetBuffer[ NTP_PACKET_SIZE]; // Буфер для хранения входящих
                                     // и исходящих пакетов

// Экземпляр объекта EthernetUDP для обмена сообщениями по протоколу UPD
EthernetUDP Udp;

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

```
  // Инициализируем Ethernet и UDP
  if (Ethernet.begin(mac) == 0)
  {
    Serial.println("Failed to configure Ethernet using DHCP");
                // Не удалось настроить Ethernet по DHCP
    while(1); // Останов
  }
  Udp.begin(localPort);
}

void loop()
{
  sendNTPpacket(timeServer); // Отправляем NTP-пакет серверу времени
                             // Ждем, будет ли ответ
  delay(1000);
  if ( Udp.parsePacket() )
  {
    Udp.read(packetBuffer,NTP_PACKET_SIZE); // Считываем пакет в буфер

    // Временная метка начинается с байта 40. Преобразовываем 4 байта
    // в целое число типа long
    unsigned long hi = word(packetBuffer[40], packetBuffer[41]);
    unsigned long low = word(packetBuffer[42], packetBuffer[43]);
    unsigned long secsSince1900 = hi << 16 | low; // Это время NTP
    unsigned long secsSince1900 = hi << 16 | low; // Это время NTP
    unsigned long secsSince1900 = hi << 16 | low; // Это время NTP
              // (количество секунд, прошедших с 1 января 1900 г.)

    Serial.print("Seconds since Jan 1 1900 = " );
              // Количество секунд, прошедших с 1 января 1900 г. =
    Serial.println(secsSince1900);

    Serial.print("Unix time = "); // UNIX-время =
    // Время Unix начинается 1 января 1970 г.
    const unsigned long seventyYears = 2208988800UL;
    // Отнимаем 70 лет
    unsigned long epoch = secsSince1900 - seventyYears;
    Serial.println(epoch);      // Отображаем UNIX-время

    // Выводим на экран час, минуты и секунды
    // UTC – это время по гринвичскому меридиану  (GMT)
    Serial.print("The UTC time is ");
    // Выводим на экран час (86400 – количество секунд в дне)
    Serial.print((epoch % 86400L) / 3600);
    Serial.print(':');
    if ( ((epoch % 3600) / 60) < 10 )
```

```
    {
      // Вставляем ведущий ноль для первых 10 минут каждого часа
      Serial.print('0');
    }
    // Выводим на экран минуты (3600 — количество минут в дне)
    Serial.print((epoch % 3600) / 60);
    Serial.print(':');
    if ( (epoch % 60) < 10 )
    {
      // Вставляем ведущий ноль для первых 10 секунд каждой минуты
      Serial.print('0');
    }
    Serial.println(epoch %60); // Выводим на экран секунды
  }
  // Ожидаем 10 секунд, прежде чем повторять запрос
  delay(10000);
}

// Посылаем NTP-запрос серверу времени по указанному адресу
unsigned long sendNTPpacket(IPAddress& address)
{
  memset(packetBuffer, 0, NTP_PACKET_SIZE); // Обнуляем значения всех байтов буфера
  // Инициализируем значения, требуемые для создания NTP-запроса
  packetBuffer[0] = B11100011; // LI, Версия, Режим
  packetBuffer[1] = 0;     // Страта
  packetBuffer[2] = 6;     // Максимальный интервал между сообщениями в секундах
  packetBuffer[3] = 0xEC; // Точность часов
  // Байты 4-11 требуются для Root Delay и Dispersion, и были обнулены
  // функцией memset()
  packetBuffer[12] = 49;  // Четыре байта контрольного ID
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;

  // Всем полям NTP были присвоены значения
  // Теперь можно отправлять пакет запроса временной метки
  Udp.beginPacket(address, 123); // Запросы NTP посылаются на порт 123
  Udp.write(packetBuffer,NTP_PACKET_SIZE);
  Udp.endPacket();
}
```

Обсуждение работы решения и возможных проблем

Протокол NTP предназначен для синхронизации времени через сообщения, передаваемые по сети Интернет. Серверы NTP предоставляют время в виде количества секунд, прошедших с 1 января 1900 года. Они используют время UTC (Universal Time Coordinated, всеобщее скоординированное время), которое заменило время по

гринвичскому меридиану, и не принимают во внимание часовые пояса и летнее время.

Серверы NTP обмениваются сообщениями, используя протокол UDP (подробно об этом протоколе рассказано в *разд. 15.3*). Сообщение NTP составляется функцией `sendNTPpacket()`, код которой вряд ли потребует изменений. Функции передается адрес сервера NTP. Для своих проектов вы можете использовать адрес сервера NTP, используемый в этом решении, или же какой-либо другой, который можно найти, выполнив поиск в Интернете по ключевой фразе `NTP address`. Дополнительную информацию о назначении полей сообщения NTP вы найдете в документации на этот протокол (**http://www.ntp.org/**).

Сервер NTP отвечает сообщением фиксированного формата, в котором информация о текущем времени начинается с байта 40 и занимает 4 байта. Эти четыре байта представляют 32-разрядное двоичное значение (беззнаковое целое число типа `long`), равное количеству секунд, прошедших с 1 января 1900 года. Это значение, а также его эквивалент в виде времени UNIX выводятся на экран. Для преобразования текущего времени из формата, возвращаемого сервером NTP, в более удобочитаемый формат с часами, минутами и секундами и днями, месяцами и годами можно использовать библиотеку TimeLib среды Arduino IDE (подробно о работе с этой библиотекой рассказано в *главе 12*). В листинге 15.33 приводится исходный код скетча, который выводит на экран время в формате `ЧЧ:ММ:СС День Число Месяц ГГГГ`.

Листинг 15.33. Скетч для вывода на экран времени NTP в формате ЧЧ:ММ:СС День Число Месяц ГГГГ

```
/*
* Скетч Time_NTP
* Демонстрирует синхронизацию времени со временем сервера NTP
* Используем библиотеки Arduino TimeLib и Ethernet
*/

#include <TimeLib.h> // См. в  тексте
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUDP.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 44 }; // Укажите здесь действительный
                                 // IP-адрес (или используйте DHCP)

unsigned int localPort = 8888; // Локальный порт, на котором
                               // ожидаются UDP-сообщения

IPAddress timeServer(192, 43, 244, 18); // Сервер NTP time.nist.gov

const int NTP_PACKET_SIZE= 48; // Временной меткой NTP являются
                               // первые 48 байтов сообщения
byte packetBuffer[ NTP_PACKET_SIZE]; // Буфер для хранения входящих и исходящих пакетов
```

```
time_t prevDisplay = 0; // Когда были отображены цифровые часы

// Экземпляр объекта EthernetUDP для обмена сообщениями по протоколу UPD
EthernetUDP Udp;

void setup()
{
   Serial.begin(9600);
   Ethernet.begin(mac,ip);
   Udp.begin(localPort);
   Serial.println("waiting for sync"); // Ожидаем синхронизацию
   setSyncProvider(getNtpTime);
   while(timeStatus()== timeNotSet);
  // Ожидаем, когда поставщик синхронизующего времени предоставит
  // текущее время
}

void loop()
{
   if( now() != prevDisplay) // Обновляем дисплей только при изменении времени
   {
      prevDisplay = now();
      digitalClockDisplay();
   }
}

void digitalClockDisplay()
{

   // Отображаем время на дисплее цифровых часов
   Serial.print(hour());
   printDigits(minute());
   printDigits(second());
   Serial.print(" ");
   Serial.print(dayStr(weekday()));
   Serial.print(" ");
   Serial.print(day());
   Serial.print(" ");
   Serial.print(monthShortStr(month()));
   Serial.print(" ");
   Serial.print(year());
   Serial.println();
}

void printDigits(int digits)
{
   // Служебная функция для дисплея цифровых часов
   // Выводит на экран предшествующее двоеточие и ведущий 0
```

```
    Serial.print(":");
    if(digits < 10)
    Serial.print('0');
    Serial.print(digits);
}

/*-------- Код NTP ----------*/

unsigned long getNtpTime()
{
    sendNTPpacket(timeServer); // Отправляем NTP-пакет серверу времени
    delay(1000);
    if ( Udp.parsePacket() )
    {
        Udp.read(packetBuffer,NTP_PACKET_SIZE); // Считываем пакет в буфер

        // Временная метка начинается с байта 40.
        // Преобразовываем 4 байта в целое число типа long
        unsigned long hi = word(packetBuffer[40], packetBuffer[41]);
        unsigned long low = word(packetBuffer[42], packetBuffer[43]);
        // Это время NTP (Количество секунд, прошедших с 1 января 1900 г.)
        unsigned long secsSince1900 = hi << 16 | low;
        // Время UNIX начинается 1 января 1970 г.
        const unsigned long seventyYears = 2208988800UL;
        // Отнимаем 70 лет
        unsigned long epoch = secsSince1900 - seventyYears;
        return epoch;
    }
    return 0; // Возвращаем 0, если не удалось получить время
}

// Посылаем NTP-запрос серверу времени по указанному адресу
unsigned long sendNTPpacket(IPAddress address)
{
    memset(packetBuffer, 0, NTP_PACKET_SIZE); // Обнуляем значения всех байтов буфера

    // Инициализируем значения, требуемые для создания NTP-запроса
    packetBuffer[0] = B11100011; // LI, Версия, Режим
    packetBuffer[1] = 0; // Страта
    packetBuffer[2] = 6; // Максимальный интервал между сообщениями
                         // в секундах
    packetBuffer[3] = 0xEC; // Точность часов
    // Байты 4-11 требуются для Root Delay и Dispersion, и были обнулены
    // функцией memset()
    packetBuffer[12] = 49; // Четырехбайтовый контрольный ID
    packetBuffer[13] = 0x4E;
    packetBuffer[14] = 49;
    packetBuffer[15] = 52;
```

```
    // Оправляем пакет запроса временной метки
    Udp.beginPacket(address, 123); // Запросы NTP посылаются на порт 123
    Udp.write(packetBuffer,NTP_PACKET_SIZE);
    Udp.endPacket();
}
```

Сообщение об ошибке: "XXXXX Was Not Declared in This Scope" (что-то не было объявлено в этой области видимости)

Этим сообщением компилятор среды Arduino IDE извещает вас, что он не знает, чем является некий элемент скетча. Если таким элементом является библиотечная функция — например, `setSyncProvider`, то это означает, что эта библиотека не была установлена или не была подключена в скетче (с помощью оператора `#include`). Информация по установке библиотеки TimeLib приводится в *главе 12*.

Дополнительная информация

Дополнительная информация по использованию библиотеки TimeLib среды Arduino IDE приводится в *главе 12*.

Подробная информация по протоколу NTP приведена на веб-сайте этого протокола (**http://www.ntp.org**).

Скетч этого решения основан на коде NTP, разработанном Jesse Jaggars (**https://oreil.ly/nlcFP**).

Для версий среды Arduino IDE более ранней, чем версия 1.0, следует использовать другую библиотеку UDP, которую можно загрузить здесь: **https://oreil.ly/UWXRC**.

ГЛАВА 16

Использование, модифицирование и создание библиотек

16.0. Введение

Библиотеки расширяют функциональность среды Arduino IDE. Они наращивают ее возможности, позволяя им выполнять операции, которые нельзя выполнить базовыми командами языка Arduino. Такие расширенные возможности может использовать любой скетч, в котором выполнено подключение требуемой библиотеки.

Дистрибутив программного обеспечения Arduino содержит встроенные библиотеки, предназначенные для обеспечения поддержки выполнения стандартных задач. Эти библиотеки рассматриваются в *разд. 16.1*.

Библиотеки также предоставляют людям хороший способ поделиться с другими своими наработками. Многие библиотеки сторонних разработчиков предоставляют пользователям Arduino специализированные возможности. Библиотеки часто создаются с целью упрощения работы с определенным аппаратным устройством. Для многих устройств, рассматриваемых в предыдущих главах, используются соответствующие библиотеки, облегчающие задачу взаимодействия с ними. Такие библиотеки можно установить в среду Arduino IDE с помощью Менеджера библиотек среды или же загрузить с веб-сайта GitHub.

Библиотеки также предоставляют аккуратную обертку сложного кода, облегчая работу с ним. Например, входящая в состав среды Arduino IDE библиотека Wire скрывает от пользователя большую часть сложности низкоуровневой связи (см. *главу 13*).

В этой главе мы рассмотрим, как использовать и модифицировать готовые библиотеки, а также как создавать свои собственные библиотеки.

16.1. Использование встроенных библиотек

ЗАДАЧА

Требуется использовать в скетче одну из встроенных библиотек Arduino IDE.

РЕШЕНИЕ

Для просмотра списка доступных встроенных библиотек выполните в среде Arduino IDE команду меню **Скетч | Подключить библиотеку**. Откроется список всех установленных библиотек. Первая часть списка содержит десять или более

библиотек, входящих в стандартный состав среды Arduino IDE. Вторая часть списка, отделенная от первой линией, содержит библиотеки, загруженные и установленные пользователем.

Чтобы подключить, т. е. добавить, библиотеку из этого списка к скетчу, просто щелкните мышью на требуемой библиотеке — в верхней части скетча появится следующая строка кода:

```
#include <названиеПодключеннойБиблиотеки.h>
```

В результате код, содержащийся в этой библиотеке, будет доступен для использования в скетче.

Среда Arduino IDE обновляет список доступных библиотек только при ее запуске. Чтобы установленная вручную библиотека отобразилась в этом списке, необходимо закрыть, а затем снова запустить среду Arduino IDE. При установке библиотек с помощью Менеджера библиотек перезапуск среды Arduino IDE не требуется.

На веб-странице **Libraries** (**https://oreil.ly/ZlfnW**) веб-сайта Arduino имеется документация на стандартные библиотеки среды Arduino IDE с примерами для каждой библиотеки. Подробная информация о работе с примерами скетчей Arduino IDE приведена в *главе 1*.

Среда Arduino IDE (версия 1.8.10) содержит следующие стандартные библиотеки:

- Adafruit Circuit Playground — обеспечивает поддержку платы Circuit Playground компании Adafruit, содержащей много датчиков и выходных контактов, что позволяет легко и быстро создавать прототипы устройств;
- Bridge — предназначена для поддержки снятых с производства плат Arduino Yun и TRE и шилда Yun. Обеспечивает взаимодействие между операционной системой Linux и микроконтроллерами этих плат;
- Esplora — предназначена для обеспечения поддержки снятой с производства платы Arduino Esplora, содержащей датчики, джойстик, светодиоды, зуммеры и другие вводы и выводы;
- EEPROM — обеспечивает поддержку записи и чтения данных в энергонезависимую память (ЭСППЗУ), работа с которой подробно рассматривается в *главе 18*;
- Ethernet — обеспечивает поддержку шилда Ethernet Arduino и совместимых модулей (например, модуля Ethernet FeatherWing компании Adafruit), а также плату Arduino Ethernet (работа с сетью Ethernet подробно рассматривается в *главе 15*);
- Firmata — протокол для упрощения последовательной связи и управления платой;
- GSM — обеспечивает поддержку снятого с производства шилда GSM Arduino, позволяющего подключать плату Arduino к сетям мобильной связи;
- HID — придает некоторым платам (например, Arduino Leonardo и платам с микроконтроллером SAMD) функциональность мыши или клавиатуры. Вы не

можете использовать эту библиотеку напрямую, но ее наличия требуют библиотеки Keyboard и Mouse;

- Keyboard — позволяет некоторым платам (например, Arduino Leonardo и платам с микроконтроллером SAMD) функционировать в качестве USB-клавиатуры;
- LiquidCrystal — предназначена для управления совместимыми жидкокристаллическими дисплеями, работа с которыми подробно рассматривается в *главе 11*;
- Mouse — позволяет некоторым платам (например, Arduino Leonardo и платам с микроконтроллером SAMD) функционировать в качестве USB-мыши;
- Robot Control — обеспечивает поддержку снятой с производства платы Arduino для управления роботами;
- Robot IR Remote — обеспечивает поддержку ИК дистанционного управления снятого с производства робота Arduino Robot;
- Robot Motor — обеспечивает поддержку снятой с производства платы Arduino Robot Motor;
- SD — предназначена для считывания и записи карт SD. Использует дополнительные периферийные устройства;
- Servo — предназначена для управления сервоприводами, работа с которыми рассматривается в *главе 8*;
- SoftwareSerial — позволяет создавать дополнительные программные последовательные порты;
- SpacebrewYun — предназначена для работы со снятой с производства платой Arduino Yun для обеспечения связи на основе WebSocket;
- SPI — обеспечивает аппаратную поддержку протоколов Ethernet и SPI, работа с которыми рассматривается в *главе 13*;
- Stepper — предназначена для работы с шаговыми двигателями, работа с которыми рассматривается в *главе 8*;
- Temboo — используется для организации взаимодействия платы Arduino с платформой Temboo (**https://temboo.com**), предназначенной для подключения к интерфейсам API, базам данных и программным утилитам;
- TFT — предназначена для работы со снятым с производства ЖК-дисплеем для Arduino. Сторонние платы и модули обычно доступны вместе с пользовательскими библиотеками;
- WiFi — обеспечивает поддержку шилда Wi-Fi для Arduino, который в настоящее время снят с производства и заменен платами Arduino со встроенными возможностями Wi-Fi, а также платами и модулями сторонних производителей;
- Wire — предназначена для поддержки устройств I^2C, подключаемых к плате Arduino. Интерфейс I^2C подробно рассматривается в *главе 13*;

Следующие две библиотеки входили в состав более ранних версий среды Arduino IDE, чем версия 1.0, но не входят в более поздние версии среды:

- Matrix — поддерживает работу со светодиодными матрицами, которые подробно рассматриваются в *главе 7*;
- Sprite — позволяет использовать спрайты на светодиодной матрице.

Обсуждение работы решения и возможных проблем

Библиотеки, предназначенные для взаимодействия с конкретными аппаратными блоками микроконтроллера Arduino, работают только с предопределенными контактами. Примерами таких библиотек являются библиотеки Wire и SPI. Другие библиотеки позволяют пользователю выбирать контакты, обычно информируя об этом в функции `Setup()`. Примерами таких библиотек могут служить библиотеки Servo, LiquidCrystal и Stepper. Информация по конфигурированию используемой библиотеки приводится в ее документации.

Подключение библиотеки к скетчу добавляет в него содержащийся в ней код. Соответственно, размер вашего скетча, о котором сообщается в конце процесса компиляции, будет несколько увеличен относительно его первоначального значения. Однако компилятор Arduino достаточно сообразительный, чтобы включать в скетч только тот код, который этот скетч фактически использует из библиотеки, поэтому вам можно не беспокоиться по поводу напрасной траты памяти на методы, которые в скетче не используются.

Библиотеки, входящие в стандартную установку Arduino IDE (а также многие библиотеки сторонних разработчиков), содержат скетчи примеров, показывающие, как использовать библиотеку. Чтобы получить доступ к этим скетчам примеров, нужно выполнить команду меню среды Arduino IDE **Файл | Примеры**.

Дополнительная информация

Более подробная информация по библиотекам Arduino приводится на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/1qFrt**).

16.2. Установка библиотек сторонних разработчиков

ЗАДАЧА

В разрабатываемом скетче требуется использовать библиотеку Arduino стороннего разработчика, не входящую в состав стандартной поставки среды Arduino IDE.

РЕШЕНИЕ

Прежде всего нужно проверить, есть ли требуемая библиотека в списке доступных библиотек Менеджера библиотек. Отройте окно Менеджера библиотек, выполнив команду меню среды Arduino IDE **Инструменты | Управление библиотеками**, и введите в поле поиска название требуемой библиотеки (или название компонента, для работы с которым она предназначена, — требуемую библиотеку часто можно

найти именно таким образом). Выберите библиотеку, щелкнув на ней мышью, а затем нажмите кнопку **Установка** (рис. 16.1). Библиотеку можно будет использовать сразу же после завершения процесса установки.

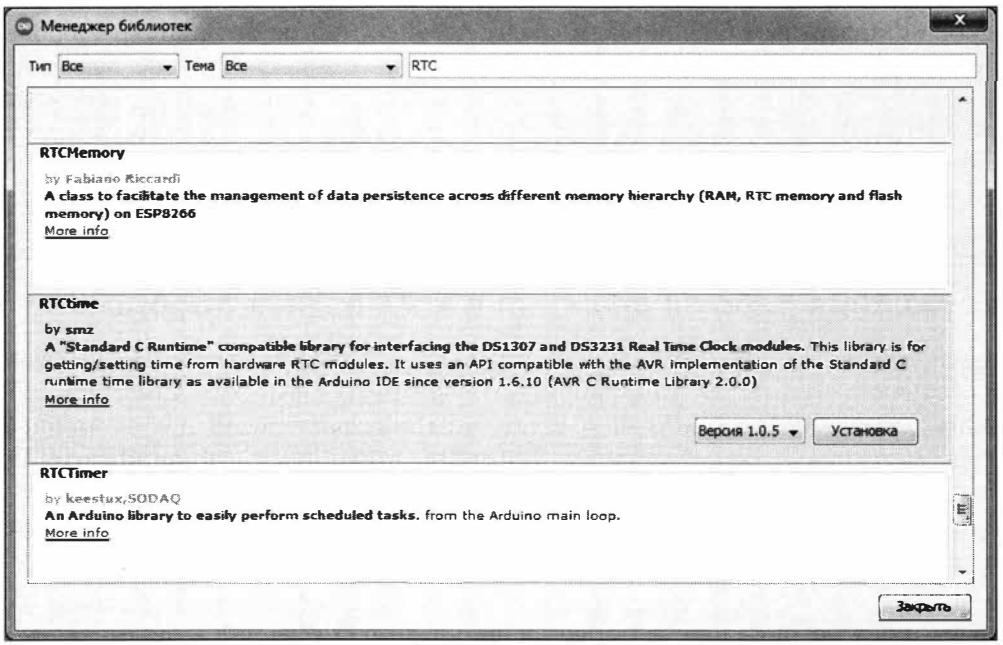


Рис. 16.1. Установка библиотеки стороннего разработчика с помощью Менеджера библиотек среды Arduino IDE

Если нужная библиотека отсутствует в списке Менеджера библиотек, ее следует сначала загрузить. Если библиотека предоставляется на веб-сайте GitHub, внимательно изучите инструкции по загрузке в ее файле README. В большинстве случае ее можно загрузить со вкладки **Releases**, расположенной непосредственно над списком файлов, или нажав кнопку **Code** справа над списком файлов и выбрав в открывшемся списке опцию **Download Zip**. Если сторонний разработчик предоставляет свою библиотеку каким-либо иным способом, она также часто будет упакованной в ZIP-архив. В результате распаковки загруженного архива создастся папка с таким же названием, что и название библиотеки. Эту папку нужно поместить в папку libraries, находящуюся в папке, в которой сохраняются разрабатываемые скетчи Arduino. Узнать местонахождение этой папки можно в окне **Настройки** (выполнив команду меню среды Arduino IDE **Arduino | Настройки** для систем macOS или **Файл | Настройки** для систем Windows или Linux). Путь к папке скетчей будет указан в поле **Размещение папки скетчей** в верхней части окна. Откройте эту папку в проводнике Windows (или в файловом менеджере Finder для систем macOS) и скопируйте во вложенную в нее папку libraries распакованную папку библиотеки. Если папка libraries отсутствует в папке скетчей Arduino, создайте ее самостоятельно.

Чтобы сделать установленную библиотеку доступной, надо закрыть среду Arduino IDE и снова запустить ее. Среда Arduino IDE проверяет содержимое папки libraries

только при запуске. Если теперь выполнить команду меню среды Arduino IDE **Скетч | Подключить библиотеку**, то в разделе **Установленные библиотеки** (Contributed) открывшегося списка доступных библиотек должна будет присутствовать только что установленная библиотека.

Подобно стандартным библиотекам среды Arduino, многие библиотеки сторонних разработчиков также содержат примеры скетчей, демонстрирующие реализацию наиболее распространенных задач в своей области. Получить доступ к ним можно, выполнив команду меню **Файл | Примеры**, — примеры установленной библиотеки будут находиться в разделе с ее названием, расположенном между разделом общих примеров среды Arduino IDE и разделом примеров библиотек стандартной установки среды.

Обсуждение работы решения и возможных проблем

Для платформы Arduino предлагается огромное количество библиотек сторонних разработчиков. Многие из этих библиотек выполнены с высоким качеством, обеспечены регулярной поддержкой и дополнены содержательной документацией и примерами скетчей. На веб-странице **Libraries** (**https://www.arduinolibraries.info**) сайта Arduino приводится большой и регулярно обновляемый список всех доступных библиотек. Хорошим источником библиотек также является и веб-страница Arduino Playground (**https://oreil.ly/kgfit**).

При выборе библиотек рекомендуется отдавать предпочтение библиотекам, обладающим качественной документацией и примерами. О качестве интересующей вас библиотеки можно узнать, проверив наличие ее обсуждений на форумах по Arduino и просмотрев эти обсуждения. При использовании с последними версиями среды Arduino IDE библиотек, предназначенных для работы с ранними версиями среды, могут возникать проблемы, поэтому вам может потребоваться рассмотреть большой объем справочного материала (некоторые ветви обсуждений популярных библиотек содержат сотни записей), чтобы найти информацию по использованию старой библиотеки с самыми последними версиями среды Arduino IDE.

Если меню **Примеры** не содержит примеров для требуемой библиотеки или при попытке использовать библиотеку выводится сообщение об ошибке **Library not found** (Не удалось найти библиотеку), проверьте правильность размещения папки библиотеки и написания ее названия. Папка библиотеки *НазваниеБиблиотеки* (например ConfigurableFirmata) должна содержать файл *НазваниеБиблиотеки*.h с точно таким же именем, включая написание заглавных и прописных букв. Для приведенного примера это должен быть файл ConfigurableFirmata.h. Также проверьте наличие в папке библиотеки всех других необходимых файлов.

16.3. Модифицирование библиотеки

ЗАДАЧА

Требуется модифицировать поведение существующей библиотеки — например, расширить ее возможности. Пусть вам недостаточно шести поддерживаемых сиг-

нальных событий библиотеки TimeAlarms (см. *разд. 12.5*), и вы хотите увеличить их количество.

РЕШЕНИЕ

Стандартная функциональность библиотек Time и TimeAlarms подробно рассматривается в *главе 12* и, в частности, в *разд. 12.5*. Эти библиотеки можно установить стандартным способом с помощью Менеджера библиотек. В случае проблем с нахождением в списке библиотеки Time попробуйте выполнить ее поиск по ключевому слову `timekeeping`.

Установив библиотеки Time и TimeAlarms, загрузите в свою плату Arduino с микроконтроллером AVR скетч из листинга 16.1, который попытается создать семь сигнальных событий. Это на одно сигнальное событие больше, чем поддерживается этими библиотеками (т. е. поддерживается для плат с микроконтроллером AVR, поскольку платы ESP8266 и платы с микроконтроллером ARM поддерживают более чем шесть сигнальных событий). Каждое сигнальное событие просто выводит свой номер задачи в окно монитора порта.

Листинг 16.1. Скетч для создания семи сигнальных событий для плат AVR

```
/*
* Скетч multiple_alarms
* Использует на один таймер больше, чем поддерживается стандартной
  библиотекой, - для поддержки более чем 6 сигнальных событий нужно
  отредактировать заголовочный файл библиотеки
*/

#include <TimeLib.h>
#include <TimeAlarms.h>

int currentSeconds = 0;

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

   // Создаем 7 задач сигнальных событий
   Alarm.timerRepeat(1, repeatTask1);
   Alarm.timerRepeat(2, repeatTask2);
   Alarm.timerRepeat(3, repeatTask3);
   Alarm.timerRepeat(4, repeatTask4);
   Alarm.timerRepeat(5, repeatTask5);
   Alarm.timerRepeat(6, repeatTask6);
   Alarm.timerRepeat(7, repeatTask7); // 7-й таймер повторён
}
```

```
void repeatTask1()
{
   Serial.print("task 1 "); // Задача 1
}

void repeatTask2()
{
   Serial.print("task 2 ");
}

void repeatTask3()
{
   Serial.print("task 3 ");
}

void repeatTask4()
{
   Serial.print("task 4 ");
}

void repeatTask5()
{
   Serial.print("task 5 ");
}

void repeatTask6()
{
   Serial.print("task 6 ");
}

void repeatTask7()
{
   Serial.print("task 7 ");
}

void loop()
{
   if(second() != currentSeconds)
   {
      // Отображаем время каждую секунду
      // При активировании сигнального события в окне монитора порта
      // выводится номер его задачи.
      Serial.println();
      Serial.print(second());
      Serial.print("->");
      currentSeconds = second();
      Alarm.delay(1); // Вызывается для обработки сигнальных событий
   }
}
```

Теперь откройте окно монитора порта и наблюдайте за выводимыми в нем данными. После девяти секунд в нем должна отобразиться информация, показанная в листинге 16.2.

Листинг 16.2. Результаты работы скетча из листинга 16.1

```
1->task 1
2->task 1 task 2
3->task 1 task 3
4->task 1 task 2 task 4
5->task 1 task 5
6->task 1 task 2 task 3 task 6
7->task 1
8->task 1 task 2 task 4
9->task 1 task 3
```

Задача, запланированная на сигнальное событие седьмой секунды, не активировалась, поскольку стандартная библиотека предоставляет только шесть объектов таймера.

Количество таймеров можно увеличить, соответственно отредактировав библиотеку. Откройте папку libraries, которая находится в папке для скетчей Arduino.

Узнать местонахождение этой папки можно в поле **Размещение папки скетчей** в верхней части окна **Настройки**, которое надо открыть, выполнив команду среды Arduino IDE **Файл | Настройки** (для систем Windows и Linux) или **Arduino | Настройки** (для систем macOS).

В папке libraries откройте вложенную папку TimeAlarms, а в ней найдите и откройте заголовочный файл TimeAlarms.h (заголовочные файлы рассматриваются более подробно в *разд. 16.4*). Для редактирования файла можно использовать любой текстовый редактор — например, Блокнот в Windows или TextEdit в macOS.

Верхняя часть файла TimeAlarms.h должна содержать код, показанный в листинге 16.3.

Листинг 16.3. Фрагмент кода заголовочного файла TimeAlarms.h

```
#ifndef TimeAlarms_h
#define TimeAlarms_h

#include <Arduino.h>
#include "TimeLib.h"

#if defined(__AVR__)
#define dtNBR_ALARMS 6 // max is 255
#else
#define dtNBR_ALARMS 12 // assume non-AVR has more memory
#endif
```

Максимальное количество сигнальных событий (таймеров) определяется значением, присвоенным константе `dtNbr_ALARMS` в строке кода:

```
#define dtNBR_ALARMS 6
```

Отредактируйте эту строку следующим образом:

```
#define dtNBR_ALARMS 7
```

после чего сохраните файл.

Снова загрузите скетч из листинга 16.1 в свою плату Arduino и откройте окно монитора порта. На этот раз в нем должна отображаться информация, показанная в листинге 16.4.

Листинг 16.4. Результаты исполнения скетча из листинга 16.1 после редактирования заголовочного файла TimeAlarms.h

```
1->task 1
2->task 1 task 2
3->task 1 task 3
4->task 1 task 2 task 4
5->task 1 task 5
6->task 1 task 2 task 3 task 6
7->task 1 task 7
8->task 1 task 2 task 4
9->task 1 task 3
```

Как можно видеть, теперь задача 7 активируется после 7 секунд.

Обсуждение работы решения и возможных проблем

Предоставляемые библиотеками возможности представляют собой компромисс между используемыми библиотекой ресурсами и ресурсами, доступными для остальной части скетча. И в случае необходимости этот баланс можно изменить. Предположим, например, что нам нужно ограничить объем памяти, используемый библиотекой Serial, чтобы иметь больше памяти RAM для работы другого кода скетча. Или же наоборот, надо позволить библиотеке использовать больший объем памяти. Разработчики библиотек обычно создают их для решения типичных задач, но если для вашего приложения требуются возможности, не предоставленные разработчиком библиотеки, не исключено, что эти возможности можно создать, модифицировав библиотеку соответствующим образом.

В только что рассмотренном примере стандартная версия библиотеки TimeAlarms выделяет в памяти RAM место для шести сигнальных событий. Каждое из этих событий использует около десяти байтов, которые резервируются, даже если предусмотрено меньше, чем все шесть возможных событий. Количество сигнальных событий задается в заголовочном файле библиотеки (в нашем случае — в заголовочном файле TimeAlarms.h, расположенном в папке TimeAlarms).

В библиотеке TimeAlarms максимальное количество сигнальных событий задается оператором `#define`. Отредактировав этот оператор и сохранив заголовочный файл, при повторном компилировании (загрузке) скетча, мы задали новое значение максимального количества сигнальных событий.

Иногда операторы `#define` (или константы) используются для задания таких характеристик платы, как рабочая частота, и при использовании заголовочного файла с платой, работающей на другой частоте, можно получить неожиданные результаты. Эта проблема решается редактированием заголовочного файла с указанием в нем правильной рабочей частоты для используемой платы.

Если по какой-либо причине после редактирования заголовочного файла библиотека прекращает работать должным образом, ее можно загрузить повторно, возвратив всю библиотеку в исходное состояние.

Дополнительная информация

В *разд. 16.4* содержится более подробная информация по расширению функциональности библиотек.

16.4. Создание собственных библиотек

ЗАДАЧА

Требуется создать свою библиотеку. Библиотеки предоставляют удобный способ использования одного и того же кода во многих скетчах, а также дают другим разработчикам возможность использовать ваш код.

РЕШЕНИЕ

Библиотека представляет собой набор методов и переменных, объединенных в формате, предоставляющем пользователю стандартный способ доступа к этим функциям и переменным.

Большинство библиотек Arduino создаются в виде класса. Те из вас, кто знаком с языком C++ или Java, должны знать, что такое *класс*. Однако библиотеки можно также создавать и без использования классов, и здесь мы увидим, как это сделать.

Итак, давайте модифицируем скетч решения из *разд. 7.1*, чтобы включить функцию `BlinkLED()` в библиотеку.

Схема подключения светодиодов приводится на рис. 7.2, а описание ее работы — в *разд. 7.1*. Создаваемая нами библиотека будет содержать функцию `blinkLED()` из этого решения. В листинге 16.5 приводится код скетча blinkLibTest, с помощью которого будет выполняться тестирование созданной библиотеки.

Листинг 16.5. Скетч для тестирования созданной библиотеки

```
/*
* Скетч blinkLibTest
*/
```

```
#include "blinkLED.h"

const int firstLedPin = 3; // Контакты для подключения светодиодов
const int secondLedPin = 5;
const int thirdLedPin = 6;

void setup()
{
   pinMode(firstLedPin, OUTPUT);  // Задаем выходной режим работы для
                                  // контактов светодиодов
   pinMode(secondLedPin, OUTPUT);
   pinMode(thirdLedPin, OUTPUT);
}

void loop()
{
   // Мигаем каждым светодиодом один раз в 1,000 мс (1 секунду)
   blinkLED(firstLedPin, 1000);
   blinkLED(secondLedPin, 1000);
   blinkLED(thirdLedPin, 1000);
}
```

Функцию `blinkLED()` нужно удалить из скетча решения, приведенного в *разд. 7.1*, и поместить в отдельный файл с названием blinkLED.cpp, как показано в листинге 16.6 (файлы *.cpp рассматриваются более подробно далее — в *разд.* «*Обсуждение работы решения и возможных проблем*» этого раздела).

Листинг 16.6. Код библиотеки blinkLED

```
/* blinkLED.cpp
* Простая библиотека для мигания светодиодом с периодом в миллисекундах
*/

#include "Arduino.h" // Для версий среды Arduino IDE более ранних,
                     // чем версия 1.0, используйте файл Wprogram.h
#include "blinkLED.h"

// Мигаем светодиодом на этом контакте в течение duration миллисекунд
void blinkLED(int pin, int duration)
{
   digitalWrite(pin, HIGH); // Включаем светодиод
   delay(duration);
   digitalWrite(pin, LOW); // Выключаем светодиод
   delay(duration);
}
```

Большинство библиотек разрабатывается программистами, которые используют для этого свою среду разработки, но их с таким же успехом можно создавать и в любом обычном текстовом редакторе типа Блокнот.

А теперь создайте заголовочный файл blinkLED.h и скопируйте в него следующий код:

```
/*
* blinkLED.h
* Заголовочный файл для библиотеки BlinkLED
*/

#include "Arduino.h"

void blinkLED(int pin, int duration); // Прототип функции
```

Обсуждение работы решения и возможных проблем

Наша библиотека будет называться blinkLED и храниться в папке libraries (см. *разд. 16.2*). Создайте в этой папке вложенную папку blinkLED и переместите в нее файлы blinkLED.h и blinkLED.cpp. Затем создайте в папке blinkLED вложенную папку examples, а в ней — вложенную папку blinkLibTest. Создайте из скетча для тестирования библиотеки (см. листинг 16.5) файл с именем blinkLibTest.ino и поместите его в папку blinkLibTest. Путь к этому файлу должен выглядеть так: examples/blinkLibTest/blinkLibTest.ino.

В результате мы переместили функцию `blinkLED()` скетча решения из *разд. 7.1* в файл библиотеки blinkLED.cpp (расширением cpp обозначаются файлы с исходным кодом, созданным на языке C++, — C plus plus).

Используемые в документации библиотек Arduino термины *функция* и *метод* обозначают блоки кода — такие как `blinkLED()`. Термин *метод* служит для обозначения функциональных блоков классов. Оба термина обозначают именованные функциональные блоки, доступ к которым предоставляется посредством библиотек.

Файл blinkLED.cpp содержит исходный код функции `blinkLED()`, который идентичен коду этой функции в скетче из листинга 7.1, за исключением следующих двух строк в начале кода:

```
#include "Arduino.h" // Подключение заголовочного файла Arduino.h
#include "blinkLED.h"
```

Оператор `#include "Arduino.h"` требуется для библиотек, использующих любые функции или константы языка Arduino. При отсутствии этого оператора компилятор будет выдавать сообщения об ошибке для всех функций, используемых в скетче.

Заголовочный файл *Arduino.h* был добавлен в версии 1.0 среды Arduino IDE, заменив заголовочный файл *WProgram.h*, используемый в более ранних версиях среды. При работе с такими более ранними версиями среды Arduino IDE можно использовать следующую конструкцию, чтобы подключать правильный заголовочный файл:

```
#if ARDUINO >= 100
  #include "Arduino.h // для версий 1.0 и более поздних
```

```
#else
  #include "WProgram.h" // для более ранних версий
#endif
```

Следующая строка кода: `#include "blinkLED.h"` — содержит определение функции (которое также называется *прототипом функции*) для нашей библиотеки. Компилятор Arduino автоматически создает прототипы для всех функций в скетче при его компилировании, но не создает прототипов для функций, содержащихся в библиотеках, поэтому при создании библиотеки необходимо самому создать заголовочный файл, содержащий эти прототипы. При подключении библиотеки к скетчу в него добавляется именно этот заголовочный файл (см. *разд. 16.1*).

Каждая библиотека должна иметь файл, в котором объявляются названия предоставляемых функций. Такой файл называется *заголовочным* (header file) или *включаемым* (include file), а его название имеет формат *НазваниеБиблиотеки.h*. В рассматриваемом примере заголовочный файл называется *blinkLED.h* и размещается в той же папке, что и файл *blinkLED.cpp*.

Содержимое заголовочного файла для нашей библиотеки очень простое — объявление одной функции:

```
void blinkLED(int pin, int duration); // Прототип функции
```

Это объявление почти такое же, как и объявление этой функции в файле blinkLED.cpp:

```
void blinkLED(int pin, int duration)
```

Однако разница между этими двумя объявлениями функции хотя и тонкая, но критическая. Объявление прототипа функции в заголовочном файле завершается точкой с запятой. Это сообщает компилятору, что это просто объявление формата для функции, но не ее кода. А отсутствие завершающей точки с запятой в объявлении функции в файле исходного кода blinkLED.cpp сообщает компилятору, что это собственно исходный код функции.

Библиотеки могут содержать несколько заголовочных файлов и несколько файлов исходного кода. Но они должны иметь минимум один заголовочный файл, название которого должно быть идентичным названию библиотеки. Название именно этого файла вставляется вверху скетча при подключении библиотеки.

Более подробную информацию по использованию заголовочных файлов и файлов *.cpp для создания модулей кода можно получить, прочитав какую-либо хорошую книгу по языку C++ (в *разд. «Дополнительная информация»* этого раздела рекомендуется несколько таких весьма популярных книг).

Разместив файлы blinkLED.cpp, blinkLED.h и blinkLibTest.ino в соответствующие вложенные папки в папке libraries, закройте среду разработки Arduino IDE, а затем снова запустите ее. Структура задействованных папок и файлов должна выглядеть следующим образом:

```
libraries/
└── blinkLED/
    ├── blinkLED.cpp
    ├── blinkLED.h
    └── examples/
        └── blinkLibTest/
            └── blinkLibTest.ino
```

Среда Arduino IDE обновляет список доступных библиотек только при ее запуске. Чтобы установленная вручную библиотека отобразилась в этом списке, необходимо закрыть, а затем снова запустить среду Arduino IDE. И хотя среду Arduino IDE нужно перезапустить при исходном добавлении библиотеки в папку *libraries*, впоследствии — после модифицирования библиотеки — перезапуск не требуется.

Выполнив команду меню **Файл | Примеры (Примеры из пользовательских библиотек) | blinkLED | blinkLibTest**, откройте в окне Arduino IDE тестовый скетч blinkLibTest. Загрузите этот скетч в свою плату Arduino, и подключенные к ней светодиоды должны начать мигать так же, как и при исполнении скетча решения из *разд. 7.1*.

Добавить новую функциональность в существующую библиотеку не составляет никаких проблем. Предположим, например, что вы хотите добавить в нее несколько постоянных, наиболее часто используемых значений задержки, чтобы пользователи библиотеки могли применять описательные названия констант вместо значений в миллисекундах.

Эта задача решается добавлением в заголовочный файл библиотеки трех строк кода со значениями констант, которые обычно размещаются непосредственно перед кодом объявления прототипа функции:

```
// Константы для периода мигания светодиодов
const int BLINK_SHORT = 250;
const int BLINK_MEDIUM = 500;
const int BLINK_LONG = 1000;

void blinkLED(int pin, int duration); // Прототип функции
```

Затем надо модифицировать код в функции `loop()` скетча примера, как показано в листинге 16.7. Загрузив модифицированный скетч в плату, мы увидим, что каждый светодиод мигает с другой частотой.

Листинг 16.7. Модифицированный код функции `loop()` для мигания светодиодами с разной частотой

```
void loop()
{
   blinkLED(firstLedPin, BLINK_SHORT);
   blinkLED(secondLedPin, BLINK_MEDIUM);
   blinkLED(thirdLedPin, BLINK_LONG);
}
```

Также легко добавляются в библиотеку и новые функции. В листинге 16.8 приводится код главного цикла `loop()`, который мигает каждым светодиодом заданное для него количество раз.

Листинг 16.8. Код цикла `loop()` для мигания каждым светодиодом в течение заданного для него периода

```
void loop()
{
   blinkLED(firstLedPin, BLINK_SHORT, 5);     // Мигает 5 раз
   blinkLED(secondLedPin, BLINK_MEDIUM, 3);   // Мигает 3 раза
   blinkLED(thirdLedPin, BLINK_LONG);         // Мигает один раз
}
```

Чтобы добавить эту функцию в библиотеку, в файл blinkLED.h нужно добавить ее прототип, как показано в листинге 16.9.

Листинг 16.9. Добавление прототипа новой функции в заголовочный файл

```
/*
* blinkLED.h
* Заголовочный файл для библиотеки blinkLED
*/

#include "Arduino.h"

// Константы для периода мигания светодиодов
const int BLINK_SHORT = 250;
const int BLINK_MEDIUM = 500;
const int BLINK_LONG = 1000;

void blinkLED(int pin, int duration);

// Определение новой функции для мигания заданного количества раз
void blinkLED(int pin, int duration, int repeats);
```

А в файл blinkLED.cpp добавляем исходный код функции, как показано в листинге 16.10.

Листинг 16.10. Исходный код функции мигания светодиодами заданное количество раз

```
/*
* blinkLED.cpp
* Простая библиотека для мигания светодиодом с периодом в миллисекундах
*/
```

```
#include "Arduino.h"
#include "blinkLED.h"

// Мигаем светодиодом на этом контакте в течение duration миллисекунд
void blinkLED(int pin, int duration)
{
   digitalWrite(pin, HIGH); // Включаем светодиод
   delay(duration);
   digitalWrite(pin, LOW); // Выключаем светодиод
   delay(duration);
}

/* Функция для повторения мигания */
void blinkLED(int pin, int duration, int repeats)
{
   while(repeats)
   {
      blinkLED(pin, duration);
      repeats = repeats -1;
   }
}
```

Для созданной библиотеки можно добавить возможность выделения цветом ключевых слов при просмотре исходного кода скетча в окне редактора скетчей среды IDE. Для этого нужно создать специальный файл keywords.txt. Это обычный текстовый файл, который содержит список ключевых слов и их типов: каждый тип ключевого слова выделяется другим цветом. Ключевые слова и типы разделяются табуляцией, а не просто пробелом. В листинге 16.11 приводится пример файла keywords.txt, в котором задается выделение цветом констант периода мигания светодиода.

Листинг 16.11. Файл keywords.txt для библиотеки blinkLED

```
#######################################
# Methods and Functions (KEYWORD2)
#######################################
blinkLED KEYWORD2
#######################################
# Constants (LITERAL1)
#######################################
BLINK_SHORT     LITERAL1
BLINK_MEDIUM    LITERAL1
BLINK_LONG      LITERAL1
```

Сохраните его в папке blinkLED и перезапустите среду Arduino IDE. Если теперь открыть файл скетча blinkLibTest.ino в среде Arduino IDE, то константы BLINK_SHORT, BLINK_MEDIUM и BLINK_LONG будут выделяться цветом.

Дополнительная информация

Дополнительные примеры создания библиотек рассматриваются в *разд. 16.5*.

Дополнительная информация по созданию библиотек для Arduino приводится в справочном документе «Writing a Library for Arduino» (**https://oreil.ly/vLNvx**).

Рекомендуем также следующие книги по программированию на C++:

- «Practical C++ Programming» («Практическое программирование на языке C++»), автор Steve Oualline, издательство O'Reilly;
- «C++ Primer Plus» («Язык программирования C++»), автор Stephen Prata, издательство Sams;
- «C++ Primer» («Учебник C++ для начинающих»), авторы Stanley B. Lippman, Josee Lajoie и Barbara E. Moo, издательство Addison-Wesley Professional.

16.5. Создание библиотеки, использующей другие библиотеки

ЗАДАЧА

Требуется создать библиотеку, которая использует функциональность из одной или нескольких других существующих библиотек. Например, мы хотим использовать библиотеку Wire для получения данных с игрового контроллера Wii Nunchuk.

РЕШЕНИЕ

В решении этой задачи используются функции из решения *разд. 13.6*, взаимодействующие с контроллером Wii Nunchuk с помощью библиотеки Wire.

В папке libraries создайте папку с названием Nunchuck (подробно о структуре папок библиотек рассказано в *разд. 16.4*). В этой папке создайте заголовочный файл Nunchuck.h и скопируйте в него код из листинга 16.12.

Листинг 16.12. Код заголовочного файла Nunchuck.h

```
/*
* Nunchuck.h
* Библиотека Arduino для интерфейса с игровым контроллером Wii Nunchuk
*/

#ifndef Nunchuck_included
   #define Nunchuck_included

   // Объекты для каждого поля, предоставляемого контроллером Wii Nunchuk
   enum nunchuckItems { wii_joyX, wii_joyY, wii_accelX, wii_accelY, wii_accelZ,
                        wii_btnC, wii_btnZ, wii_ItemCount };
```

```
  // Использует контакты, расположенные рядом с контактами шины I2C,
  // в качестве линий питания (плюс и "земля") для контроллера
  void nunchuckSetPowerpins();

  // Инициализирует интерфейс I2C для контроллера
  void nunchuckInit();

  // Запрашивает данные с контроллера Wii Nunchuk
  void nunchuckRequest();

  // Принимает данные с контроллера Wii Nunchuk
  // Возвращает true при успешном приеме, false в противном случае
  bool nunchuckRead();

  // Кодирует данные в формат, принимаемый большинством
  // драйверов Wii Remote
  char nunchuckDecode (uint8_t x);

  // Возвращает значение для заданного элемента
  int nunchuckGetValue(int item);

#endif
```

В этой же папке создайте файл Nunchuck.cpp и скопируйте в него код из листинга 16.13.

Листинг 16.13. Содержимое файла исходного кода Nunchuck.cpp

```
/*
* Nunchuck.cpp
* Библиотека Arduino для интерфейса с игровым контроллером Wii Nunchuk
*/

#include "Arduino.h" // Определения языка Arduino

#include "Wire.h" // Определения библиотеки Wire (I2C)
#include "Nunchuck.h" // Определения для этой библиотеки

// Константы для платы Arduino Uno(для платы Mega
// используйте контакты 19 и 18)
const int vccPin = A3; // С этих контактов берется питание
                       // (плюс и "земля") для контроллера
const int gndPin = A2;
const int dataLength = 6;         // Количество байтов в запросе
static byte rawData[dataLength]; // Массив для хранения данных,
                                  // полученных с контроллера
```

```
// Используем контакты, расположенные рядом с контактами шины I2C,
// в качестве линий питания (плюс и "земля") для контроллера
void nunchuckSetPowerpins()
{
   pinMode(gndPin, OUTPUT); // Задаем правильное состояние для контактов питания
   pinMode(vccPin, OUTPUT);
   digitalWrite(gndPin, LOW);
   digitalWrite(vccPin, HIGH);
   delay(100); // Пауза, чтобы дать питанию время стабилизироваться
}

// Инициализируем интерфейс I2C для контроллера
void nunchuckInit()
{
   Wire.begin();                  // Подключаемся к шине I2C,
                                  // как ведущее устройство
   Wire.beginTransmission(0x52);  // Передаем на устройство 0x52
   Wire.write((byte)0x40);        // Передаем адрес памяти
   Wire.write((byte)0x00);        // Передаем значение 0
   Wire.endTransmission();        // Завершаем передачу
}

// Запрашиваем данные с контроллера Wii Nunchuk
void nunchuckRequest()
{
   Wire.beginTransmission(0x52);// Передаем на устройство 0x52
   Wire.write((byte)0x00);      // Передаем один байт
   Wire.endTransmission();      // завершаем передачу
}

// Принимает данные с контроллера Wii Nunchuk
// Возвращает true при успешном приеме, false в противном случае
bool nunchuckRead()
{
   byte cnt=0;
   Wire.requestFrom (0x52, dataLength); // Запрос данных с контроллера Wii Nunchuk
   while (Wire.available ())
   {
      byte x = Wire.read();
      rawData[cnt] = nunchuckDecode(x);
      cnt++;
   }
   nunchuckRequest();       // Передаем запрос следующего пакета данных
   if (cnt >= dataLength)
      return true;          // Успешное завершение, если получены все 6 байтов
   else
      return false;         // Неудача
}
```

```
// Кодирует данные в формат, принимаемый большинством драйверов Wii Remote
char nunchuckDecode (byte x)
{
   return (x ^ 0x17) + 0x17;
}

// Возвращает значение для заданного элемента
int nunchuckGetValue(int item)
{
   if( item <= wii_accelZ)
      return (int)rawData[item];
   else if(item == wii_btnZ)
      return bitRead(rawData[5], 0) ? 0: 1;
   else if(item == wii_btnC)
      return bitRead(rawData[5], 1) ? 0: 1;
}
```

Подключите контроллер Wii Nunchuk к плате Arduino, как показано на рис. 13.12, создайте в среде Arduino IDE новый скетч, скопируйте в него код из листинга 16.14 и загрузите этот скетч в плату Arduino (если среда Arduino IDE была открыта, когда создавались файлы Nunchuck.h и Nunchuck.cpp, перезапустите ее, чтобы она обнаружила новую библиотеку). При желании этот файл можно сделать доступным в виде примера, сохранив его под названием WiichuckSerial.ino в папке .../library/Nunchuck/examples/WiichuckSerial.

Листинг 16.14. Исходный код скетча WiichuckSerial.ino для тестирования библиотеки Nunchuck

```
/*
* Скетч WiichuckSerial
* Использует библиотеку Nunchuck для передачи значений датчиков
  через последовательный интерфейс
*/

#include <Wire.h>
#include "Nunchuck.h"

void setup()
{
   Serial.begin(9600);
   nunchuckSetPowerpins();
   nunchuckInit(); // Посылаем инициализирующее квитирование
   nunchuckRead(); // Игнорируем первый раз
   delay(50);
}

void loop()
{
   nunchuckRead();
```

```
    Serial.print("H,");       // Заголовок
    for(int i=0; i < 5; i++) // Выводим на экран значения акселерометров и кнопок
    {
       Serial.print(nunchuckGetValue(wii_accelX+ i), DEC);
       Serial.write(',');
    }
    Serial.println();
    delay(20); // Время в миллисекундах между передачами
}
```

Обсуждение работы решения и возможных проблем

Включение другой библиотеки в исходный код (файл *.cpp) создаваемой библиотеки реализуется точно так же, как и подключение библиотеки в скетче: используя оператор `#include` с названием подключаемой библиотеки. Если планируется предоставлять разрабатываемую библиотеку другим пользователям, рекомендуется включить в ее документацию информацию обо всех других требующихся для нее библиотеках, особенно в том случае, если эти библиотеки не входят в состав установки среды Arduino IDE.

Основное отличие кода этой библиотеки от кода скетча из решения *разд. 13.6* состоит в добавлении заголовочного файла Nunchuck.h, который содержит прототипы функций (код скетча Arduino создает прототипы прозрачно для разработчика, тогда как для библиотек Arduino прототипы должны явно создаваться разработчиком).

Рассмотрим еще один пример создания библиотеки Arduino с использованием класса C++ для инкапсуляции функций библиотеки. Класс — это метод программирования для группирования функций и переменных, который обычно используется при создании большинства библиотек Arduino.

Создаваемую библиотеку можно использовать в качестве средства диагностики, передавая с помощью библиотеки Wire выход операторов `print()` на другую плату Arduino. Такая возможность может быть особенно полезной в случае отсутствия аппаратного последовательного порта и неприемлемости использования программного последовательного порта по причине вносимых им задержек. В рассматриваемом случае базовая функциональность класса `Print.h` Arduino для вывода на печать используется для создания новой библиотеки, которая передает выводимые данные на шину I^2C. Подключение плат Arduino друг к другу показано на рис. 13.11, а работа кода рассматривается в *разд. 13.5*.

Процесс преобразования этого кода в библиотеку выглядит так: в папке libraries создайте вложенную папку с названием i2cDebug (подробно о структуре папок библиотеки рассказано в *разд. 16.4*). В этой папке создайте заголовочный файл i2cDebug.h и скопируйте в него код из листинга 16.15.

Листинг 16.15. Заголовочный файл i2cDebug.h

```
/*
* i2cDebug.h
*/
```

```
#ifndef i2cDebug_included
#define i2cDebug_included

#include <Arduino.h>
#include <Print.h> // Класс print Arduino

class i2cDebugClass : public Print
{
   private:
      int i2cAddress;
      byte count;
      size_t write(byte c);
   public:
      i2cDebugClass();
      bool begin(int id);
};
```

В этой же папке создайте заголовочный файл i2cDebug.cpp и скопируйте в него код из листинга 16.16.

Листинг 16.16. Исходный код файла i2cDebug.cpp

```
/*
* i2cDebug.cpp
*/

#include <i2cDebug.h>

#include <Wire.h> // Библиотека Arduino для работы с интерфейсом I2C

i2cDebugClass::i2cDebugClass()
{
}

bool i2cDebugClass::begin(int id)
{
   i2cAddress = id; // Сохраняем адрес ведомого устройства
   Wire.begin(); // Подключаемся к шине I2C (адрес для ведущего
                 // устройства не обязателен)
   return true;
}

size_t i2cDebugClass::write(byte c)
{
   if( count == 0)
   {
```

```
        // Переходим сюда при первом символе передачи
        Wire.beginTransmission(i2cAddress); // Передаем полученные данные
                                            // ведомой плате
    }

    Wire.write(c);
    // Если буфер I2C заполнен или достигнут конец строки, передаем данные
    // Значение константы размера буфера BUFFER_LENGTH
    // определено в библиотеке Wire
    if(++count >= BUFFER_LENGTH || c == '\n')
    {
        // При полном буфере или символе новой строки отправляем данные
        Wire.endTransmission();
        count = 0;
    }
    return 1; // Записан один символ
}

i2cDebugClass i2cDebug; // Создаем объект I2C для отладки
```

Метод `write()` возвращает значение `size_t`, которое позволяет функции вывода на экран возвращать количество отображенных символов. Эта возможность впервые появилась в версии 1.0 среды Arduino IDE, а в более ранних версиях функции `write()` и `print()` не возвращали никаких значений. При использовании библиотеки на основе библиотек Stream или Print тип возвращаемого значения нужно изменить на `size_t`.

Создайте новый скетч и скопируйте в него код из листинга 16.17.

Листинг 16.17. Скетч для тестирования библиотеки i2cDebug

```
/*
* Скетч i2cDebug
* Скетч для тестирования библиотеки i2cDebug
*/

#include <Wire.h> // Библиотека Arduino для работы с интерфейсом I2C
#include <i2cDebug.h>

const int address = 4;    // Адрес, используемый взаимодействующими устройствами
const int sensorPin = 0; // Номер аналогового контакта для подключения датчика
int val;                  // Переменная для хранения значения, полученного с датчика

void setup()
{
    Serial.begin(9600);
    i2cDebug.begin(address);
}
```

```
void loop()
{
   // Считываем напряжение на контакте потенциометра: значение val
   // в диапазоне от 0 до 1023
   val = analogRead(sensorPin);
   Serial.println(val);
   i2cDebug.println(val);
}
```

Не забывайте о необходимости перезапустить среду Arduino IDE, чтобы она могла обнаружить новую библиотеку (подробная информация по созданию библиотек приводится в *разд. 16.4*).

Загрузите этот скетч в одну плату Arduino, в другую — скетч для ведомого устройства (листинг 13.8) и подключите платы, как показано на рис. 13.11. Затем запустите для ведомой платы монитор порта. В его окне должны отображаться данные, выводимые библиотекой, исполняющейся на ведущей плате.

Дополнительная информация

Дополнительную информацию по классам языка C++ можно найти в следующих книгах:

- «Programming Interactivity» («Интерактивное программирование»), автор Joshua Noble, издательство O'Reilly;
- «C++ Primer» («Учебник C++ для начинающих»), авторы Stanley B. Lippman, Josee Lajoie и Barbara E. Moo, издательство Addison-Wesley Professional.

16.6. Обновление библиотек сторонних разработчиков для Arduino 1.0

ЗАДАЧА

Требуется использовать библиотеку стороннего разработчика для более ранней версии языка Arduino, чем 1.0.

РЕШЕНИЕ

Для большинства библиотек нужно изменить только несколько строк кода, чтобы сделать их пригодными для использования с версией 1.0 (и более поздними версиями) языка Arduino. Например, любой из следующих операторов `#include` в заголовочных файлах этих библиотек:

```
#include "wiring.h"
#include "WProgram.h"
#include "WConstants.h"
#include "pins_arduino.h"
```

надо заменить одним оператором:

```
#include "Arduino.h"
```

Названия заголовочных файлов в операторах подключения нужно заключать в угловые скобки или кавычки.

Обсуждение работы решения и возможных проблем

При ошибке компиляции библиотек для старых версий языка Arduino в версии 1.0 обычно выдается одно или несколько следующих сообщений об ошибке:

```
source file: error: wiring.h: No such file or directory
                              (Не существует такого файла или папки)
source file: error: WProgram.h: No such file or directory
source file: error: WConstants.h: No such file or directory
source file: error: pins_arduino.h: No such file or directory
```

Идентификатор `source file:` (исходный файл) представляет полный путь файла библиотеки, который нужно обновить. Далее будет следовать список других ошибок, вызванных отсутствием указанных заголовочных файлов в версии 1.0 языка Arduino, но эти ошибки должны исчезнуть после замены старых названий заголовочных файлов названием `Arduino.h`. Определения в этих заголовочных файлах теперь включены в заголовочный файл `Arduino.h`, и решение их обновления состоит в замене подключения всех этих файлов подключением одного файла `Arduino.h`.

Если по какой-либо причине нужно использовать текущую (1.0 или более позднюю) версию среды Arduino IDE совместно с более ранними версиями среды, в скетчах можно задействовавть условные операторы подключения, как показано в листинге 16.18 (более подробно условные операторы подключения `#include` рассматриваются в *разд. 17.6*).

Листинг 16.18. Использование условных операторов подключения `#include`

```
#if ARDUINO >= 100
   #include "Arduino.h"
#else
   // Эти заголовочные файлы используются в исходной версии библиотеки
   #include "wiring.h"
   #include "pins_arduino.h"
#endif
```

Для обновления библиотек сторонних разработчиков, использующих функционал последовательного интерфейса, сетевого интерфейса Ethernet или другой функционал, синтаксис которого изменился в версии 1.0 языка Arduino, может потребоваться выполнить дополнительные модификации кода.

ГЛАВА 17

Продвинутые методы программирования и управления памятью

17.0. Введение

По мере расширения круга и сложности задач, реализуемых с помощью платформы Arduino, эффективность скетчей должна повышаться. Рассматриваемые в этой главе методы можно использовать для повышения эффективности скетчей за счет улучшения их производительности и уменьшения объема кода. В частности, эти методы могут помочь повысить скорость исполнения скетчей или уменьшить объем используемой ими памяти RAM для хранения данных. Решения, приведенные в этой главе, охватывают более глубокие технические аспекты, чем большинство остальных решений из этой книги, затрагивая детали, которые обычно скрыты под оболочкой Arduino.

Процесс сборки скетчей Arduino скрывает от пользователя сложные стороны языков C и C++, так же как и средства, используемые для преобразования скетча в машинный код, загружаемый в плату Arduino для исполнения. Но если требования производительности и ресурсов вашего скетча превышают возможности стандартной среды разработки Arduino IDE, рассматриваемые здесь примеры решений могут оказаться полезными для решения этой проблемы.

Для хранения информации плата Arduino оснащена тремя разными типами памяти[1]: программной флеш-памятью, памятью RAM (Random Access Memory — память с произвольным доступом для записи и чтения) и памятью EEPROM (Electrically Erasable Programmable Read-Only Memory — электрически стираемая программируемая память с доступом только для чтения). Каждый из этих типов памяти имеет разные характеристики и применение. Многие из примеров этой главы как раз и объясняют, что делать в тех случаях, когда одного из типов памяти недостаточно для решения поставленной задачи.

Назначение программной флеш-памяти соответствует ее названию — она используется для хранения исполняемого кода скетчей. Содержимое программной памяти может изменить только загрузчик в процессе загрузки скетча, инициализируемом

[1] В русскоязычной литературе для RAM и ROM обычно используются термины ОЗУ (оперативное запоминающее устройству) и ПЗУ (постоянное запоминающее устройство) соответственно. Использование здесь английских терминов объясняется желанием обеспечить единообразность терминологии: память RAM, память ROM и флеш-память.

программным обеспечением среды Arduino IDE, исполняющимся на компьютере. По завершении процесса загрузки содержимое программной памяти изменить нельзя. В платах Arduino объем программной памяти значительно больше объема памяти RAM, поэтому она также используется для хранения данных, которые не меняются в процессе исполнения скетча, — например, значений констант. Сам загрузчик хранится в программной памяти, занимая там определенный объем. В случае неуспешного исхода всех других попыток минимизировать объем кода, чтобы он поместился в программную память, объем свободной программной памяти можно увеличить, удалив из нее загрузчик. Но в таком случае для загрузки новых программ в плату нужно будет прибегать к помощи отдельного аппаратного программатора.

Если объем кода скетча превышает объем доступной программной памяти, попытка загрузить скетч в плату будет неуспешной и на этапе компилирования среда Arduino IDE выдаст предупреждение, что скетч слишком большой.

Память RAM служит для хранения значений задействованных в скетче переменных (включая значения переменных библиотек, используемых в скетче). Как следует из ее определения, содержимое памяти RAM можно изменять в процессе исполнения скетча. Но все это содержимое теряется при выключении питания, поскольку память RAM является *энергозависимой*. В платах Arduino объем памяти RAM намного меньший, чем объем программной флеш-памяти. Израсходование всего доступного объема этой памяти при исполнении скетча, в процессе которого переменные создаются и удаляются, вызовет ненормальное поведение скетча, т. е. сбой его исполнения.

Память EEPROM также может изменяться кодом в процессе его исполнения, но, в отличие от памяти RAM, эта память является *энергонезависимой*, т. е. сохраняет свое содержимое после выключения питания. Скорость чтения и записи памяти EEPROM значительно ниже, чем памяти RAM, поэтому эта память обычно используется только для хранения конфигурационных или иных данных, которые скетч считывает в начале исполнения, чтобы восстановить информацию с предыдущего сеанса исполнения.

Чтобы ориентироваться во всем этом, весьма полезно вникнуть в процесс подготовки средой Arduino IDE кода скетча для его загрузки в микроконтроллер платы, а также знать, как можно проверять результаты этой подготовки.

Препроцессор

Чтобы получить требуемый результат для некоторых решений этой главы, необходимо использовать *препроцессор*. Предварительная обработка является шагом первого этапа процесса сборки, в котором исходный код скетча подготавливается к компиляции. В ходе этой подготовки может выполняться поиск и замена различных функций. Строки кода с командами препроцессора обозначаются начальным символом #. Мы уже видели такие строки кода в скетчах, использующих библиотеки. В них команда `#include <название_библиотеки>` дает указание препроцессору вставить в код текущего скетча код из указанной библиотеки. Иногда поставленную

задачу можно решить, только используя препроцессор, но его синтаксис отличается от синтаксиса языков C и C++. Кроме того, он может вносить тонкие ошибки, причину которых трудно отследить, поэтому использовать его нужно с определенной толикой осторожности.

Дополнительная информация

На веб-сайте AVRfreaks (**http://www.avrfreaks.net**) приводится большой объем подробной технической информации о микроконтроллерах, используемых в разных платах Arduino.

На веб-сайте The C Preprocessor (**https://oreil.ly/kh_XU**) приведена обширная и подробная информация о препроцессоре языка C.

На веб-странице Arduino Products (**https://oreil.ly/sWbi2**) веб-сайта Arduino приведена техническая информация на все официальные платы Arduino, включая их характеристики памяти.

17.1. Процесс сборки скетчей Arduino

ЗАДАЧА

Вы хотите узнать, что происходит за кулисами при компиляции и загрузке скетча в плату Arduino.

РЕШЕНИЕ

Эту задачу можно решить, определив в окне **Настройки** (**Файл | Настройки** — для Windows и Linux, **Arduino | Настройки** — для macOS) среды Arduino IDE опции для вывода сообщений, выдаваемых компилятором при компиляции и загрузке скетча, в панель сообщений среды. Для этого в окне **Настройки** для опции **Сообщения компилятора** нужно установить галочки **Компиляция** и **Загрузка**, чтобы позволить вывод подробных сообщений при компиляции и/или загрузке скетча. Уровень подробности выводимых сообщений и предупреждений компилятора устанавливается выбором одной из опций выпадающего списка **Сообщения компилятора**: **Ничего**, **По умолчанию**, **Подробнее**, **Все**.

Обсуждение работы решения и возможных проблем

В ходе процесса компиляции и загрузки скетча выполняется много действий, результаты исполнения которых обычно не выводятся на экран, поскольку для компиляции, компоновки и загрузки скетча используются средства командной строки, работу которых среда Arduino IDE скрывает от пользователя.

Сначала файл или файлы скетча преобразовываются в файл, пригодный для обработки компилятором среды Arduino IDE (AVR-GCC). Все находящиеся в папке скетча файлы исходного кода с расширением ino (расширение по умолчанию файлов исходного кода скетчей Arduino) объединяются в один файл. Файлы с расши-

рениями c и cpp компилируются отдельно. Заголовочные файлы (с расширением h) игнорируются компилятором, если только они явно не включены в объединяемые файлы.

В начало объединенного файла добавляется оператор `#include "Arduino.h"` (`WProgram.h` — в версиях среды Arduino IDE более ранних, чем версия 1.0), чтобы подключить заголовочный файл, содержащий все определения языка Arduino, — например, функции `digitalRead()` или `analogWrite()`. В системах Windows этот файл, а также другие заголовочные файлы, находятся в папке, в которой установлено приложение среды разработки Arduino IDE, — по пути ...hardware/arduino/avr/cores/arduino.

Чтобы просмотреть содержимое этого файла в системах macOS, щелкните правой кнопкой на значке приложения Arduino IDE и выберите в контекстном меню опцию **Show Package Contents**. В открывшейся папке перейдите к папке Contents/Java, а из нее — в папку ...hardware/arduino/avr/cores/arduino.

Структура папок установки Arduino может измениться в новых выпусках приложения, поэтому сверьтесь с документацией используемого выпуска, чтобы узнать точный путь.

Чтобы скомпилированный код соответствовал требованиям языка C++, далее создаются и вставляются в него прототипы всех функций, объявленных в коде.

И наконец, добавляются значения разных констант, требуемых для микроконтроллеров используемой платы. Эти значения определяются настройками меню **Плата** и хранятся в файле boards.txt (...\Arduino\hardware\arduino\avr).

Получившийся файл затем компилируется с помощью компилятора AVR-GCC, входящего в состав установки среды разработки Arduino IDE. Исполняемый файл компилятора находится в папке ...\Arduino\hardware\arduino\avr\bin (для систем macOS эта папка включена в папку Contents/Java установки Arduino IDE, доступ к которой можно получить, как описано ранее).

Компилятор создает несколько объектных файлов (с расширением o), которые затем будут объединены с помощью компоновщика (линкера). Эти файлы сохраняются во временной папке temp пользователя, которая содержит папки с такими названиями, как, например, arduino_build_137218, содержащие все элементы сборки. Узнать местонахождение этой папки для систем Windows можно по значению переменной среды `TEMP`, выполнив в командной строке команду:

```
echo %TEMP%
```

Для систем macOS запустите оболочку Terminal и исполните в ней команду:

```
echo $TMPDIR
```

На системах Linux элементы сборки должны находиться в папке /tmp.

Объектные файлы затем компонуются в исполняемый шестнадцатеричный файл, который и будет загружен в плату. Для загрузки исполняемых файлов в память микроконтроллера Arduino используется служебная программа Avrdude.

Средства для реализации процесса сборки AVR находятся в папке ...\Arduino\hardware\tools\avr\bin\.

Одним из этих инструментов является инструмент `avr-objdump`, который может быть полезен для опытных программистов. С помощью этого инструмента можно наблюдать, как компилятор преобразовывает исходный код скетча в шестнадцатеричный код, который исполняется микроконтроллером платы. Инструмент `avr-objdump` создает листинг дизассемблирования исполняемого кода скетча, в котором объектный код дополняется соответствующим исходным кодом. Также с его помощью можно просмотреть карту памяти всех переменных, используемых в скетче. Используется этот инструмент следующим образом. Скомпилируйте скетч и перейдите в папку temp (о местонахождении которой рассказывалось ранее), содержащую папки сборки скетча arduino_build_*xxxxxx*. Одна из этих папок содержит файл с расширением ino (определить, какая именно, можно, просматривая каждую папку), используемый утилитой `avr-objdump`. Например, просмотреть содержимое файла с расширением elf для скетча Blink можно, выполнив в консоли командной строки Windows приведенную далее последовательность команд (обратите внимание на использование команды `PATH` для временного добавления пути к папке bin установки Arduino в переменную среды `PATH` для текущего сеанса исполнения приложения среды Arduino IDE). Число `706012` нужно заменить на суффикс папки arduino_build_xxxxxx, содержащей файл *.elf только что скомпилированного скетча, — в нашем случае Blink.ino.elf:

```
cd %TEMP%
cd arduino_build_706012
PATH "\Program Files (x86)\Arduino\hardware\tools\avr\bin\";%PATH%
avr-objdump -S Blink.ino.elf
```

Для систем Linux используется следующая последовательность команд:

```
cd /tmp/arduino_build_700798/
PATH=~/arduino-1.8.9/hardware/tools/avr/bin/:$PATH
avr-objdump -S Blink.ino.elf
```

А для систем macOS следующая:

```
cd $TMPDIR
cd arduino_build_97987/
PATH=/Applications/Arduino.app/Contents/Java/hardware/tools/avr/bin/:$PATH
avr-objdump -S Blink.ino.elf
```

Поскольку содержимое файла *.elf имеет весьма большой объем, удобнее вместо изучения его в окне консоли командной строки сохранить его в текстовом файле, к которому можно обратиться в любое время. Для этого надо просто перенаправить выход команды `avr-objdump` в текстовый файл следующим образом:

```
avr-objdump -S Blink.ino.elf > blink.txt
```

В команду можно также добавить опцию `-h`, чтобы вставить в ее выход заголовки разделов (что помогает для определения использования памяти):

```
avr-objdump -S -h Blink.ino.elf > blink.txt
```

Для плат с микроконтроллером иным, чем микроконтроллер AVR (например, плат с микроконтроллером ARM или ESP8266), базовые файлы Arduino (набор инструментов, заголовочные файлы и другие файлы, поддерживающие процесс компиляции для определенной аппаратной архитектуры) хранятся в другой вложенной папке папки пользователя. В системах Windows это папка %LOCALAPPDATA%\Arduino15\packages, в системах macOS — папка ~/Library/Arduino15/packages, а в системах Linux— папка ~/.arduino15/packages. Например, на момент подготовки этой книги файл утилиты `objdump` для плат с микропроцессором SAMD находится в папке ...arduino15/packages/arduino/tools/arm-none-eabi-gcc/4.8.3-2014q1/bin. Соответствующая последовательность команд для его запуска, чтобы просмотреть файл *.elf, будет следующей:

```
PATH=~/.arduino15/packages/arduino/tools/arm-none-eabi-gcc/4.8.3-2014q1/bin:$PATH
arm-none-eabi-objdump -S Blink.ino.elf
```

В случае установки нескольких базовых пакетов, включая пакеты для плат компаний Adafruit и SpartFun или плат на микроконтроллере ESP8266, папка packages может содержать несколько папок с наборами средств разработки и утилит для соответствующих типов плат. Наилучший способ определить, какая из этих папок содержит утилиты для какого типа платы, — это выполнить компиляцию скетча с выводом подробных сообщений и посмотреть полный путь к папке инструментов в выводимых сообщениях компилятора. Например, увидев на панели сообщений строку наподобие:

```
arm-none-eabi-g++-mcpu=cortex-m0plus
```

или

```
xtensa-lx106-elf-gcc -CC -E -P -DVTABLES_IN_FLASH
```

посмотрите и запомните или запишите путь к папке, содержащей ее базовый пакет средств и утилит.

Дополнительная информация

Подробная информация (на английском языке) о процессе сборки скетчей Arduino приведена на веб-странице Sketch build process (**https://oreil.ly/viClX**) сайта Arduino.

17.2. Определение объема свободной и занятой памяти RAM

ЗАДАЧА

Требуется убедиться в том, что для хранения данных скетча имеется достаточный объем памяти RAM. В случае недостатка памяти RAM скетч будет работать неправильно, но эту проблему может быть трудно заметить.

РЕШЕНИЕ

Эта задача решается с помощью скетча, исходный код которого приводится с листинге 17.1. В скетче показано, как можно определить объем свободной памяти RAM, доступной для использования скетчем. Скетч использует функцию `memoryFree()` для определения объема доступной памяти RAM микроконтроллера AVR.

Листинг 17.1. Скетч для определения доступного объема памяти RAM микроконтроллеров AVR

```
/*
* Free memory sketch for AVR
*/

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

void loop()
{
   Serial.println(memoryFree()); // Выводим в окно монитора порта объем
                                 // свободной памяти RAM
   delay(3000);
}

// Переменная, созданная процессом сборки при компилировании скетча
extern int *__brkval; // Указатель на адрес последней выделенной
                      // ячейки кучи (или 0)

// Функция, возвращающая объем свободной памяти RAM
int memoryFree()
{
   int freeValue; // Это будет самый последний объект, для которого
                  // выделена память в куче
   if((int)__brkval == 0) // Куча пустая. Используем адрес начала кучи
   {
      freeValue = ((int)&freeValue) - ((int)__malloc_heap_start);
   }
   else // Куча не пустая; используем последний адрес в куче
   {
      freeValue = ((int)&freeValue) - ((int)__brkval);
   }
   return freeValue;
}
```

Обсуждение работы решения и возможных проблем

В функции `memoryFree()` сначала объявляется переменная `freeValue`, которая помещается в кучу, поскольку она является локальной для этой функции. Управляемая память состоит из двух основных частей: *стека* и *кучи*. Стек расположен в конце свободной памяти, и при добавлении в него новых элементов занятый объем стека расширяется в направлении начала памяти. Стек используется для хранения значений локальных переменных и данных, связанных с отслеживанием прерываний и вызовов функций. По завершении исполнения функции связанные с нею данные удаляются из стека, освобождая память, и занятый объем стека сокращается. Куча расположена в начале свободной памяти, и когда скетч (или библиотека) выделяет в ней память (например, при создании объекта `String`) занятый объем кучи увеличивается по направлению к концу свободной памяти. То есть занятый объем стека увеличивается по направлению к куче, а занятый объем кучи увеличивается по направлению к стеку.

Соответственно расстояние в байтах между занятым объемом стека и занятым объемом кучи и будет объемом свободной памяти RAM. Адрес переменной `freeValue` (`&freeValue`) является адресом последнего байта свободной памяти (начало стека). Системная переменная `_brkval` содержит адрес первого байта свободной памяти (конец занятого объема кучи), если только в настоящее время куча ничего не содержит. В таком случае ее значение будет 0 и нам не будет от нее никакой пользы, поскольку область памяти перед кучей содержит много других элементов. Но адрес начала кучи содержит другая системная переменная: `_malloc_heap_start`. Поэтому, если мы знаем, что куча пустая (значение системной переменной `_brkval` = `0`), тогда нужно использовать переменную `_malloc_heap_start`.

Адрес системных переменных `_brkval` и `_malloc_heap_start` не требуется получать с помощью оператора указателя `&`, подобно тому, как это делается для переменной `freeValue`. Дело в том, что переменные `_brkval` и `_malloc_heap_start` содержат числа, уже обозначающие адрес, тогда как переменная `freeValue` содержит целочисленное значение, чей адрес мы хотим знать. Если вы задаетесь вопросом, откуда взялись переменные `_malloc_heap_start` и `_brkval`, то знайте, что они создаются в процессе компиляции скетча и доступны скетчу в виде идентификаторов. Переменная `_malloc_heap_start` доступна для всех скетчей автоматически, но для доступа к переменной `_brkval` ее нужно сначала объявить с помощью оператора объявления `extern`.

Скетч для плат с процессором ARM (листинг 17.2) несколько проще. Здесь общая идея та же, но с тем исключением, что для микроконтроллеров ARM можно использовать низкоуровневую функцию управления памятью `sbrk()`. Эта функция при вызове с аргументом `0` возвращает начальный адрес памяти RAM. Расстояние между адресом переменной `freeValue` и значением, возвращенным функцией `sbrk(0)`, и будет объемом свободной памяти RAM. Функция `sbrk()` объявляется как указатель типа `char`, поэтому в скетче ее тип нужно явно преобразовать в указатель типа `int`, чтобы ее значение можно было отнять от адреса переменной `freeValue`. Для преобразования типов вместо обычного оператора `cast [(int *)]` в скетче исполь-

зуется оператор reinterpret_cast, т. к. он позволяет выполнять преобразования, которые могут считаться небезопасными в некоторых конфигурациях.

Листинг 17.2. Скетч для определения доступного объема памяти RAM микроконтроллеров ARM

```
/*
* Free memory sketch for ARM
*/

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

void loop()
{
   Serial.println(memoryFree()); // Выводим в окно монитора порта
                                 // объем свободной памяти RAM
   delay(3000);
}

// Переменная, созданная процессом сборки при компилировании скетча
extern "C" char *sbrk(int incr); // Вызывается с аргументом 0,
        // чтобы получить начальный адрес свободной памяти RAM

// Функция, возвращающая объем свободной памяти RAM
int memoryFree()
{
   int freeValue; // Это будет самый последний объект,
                  // для которого выделена память в куче
   freeValue = &freeValue - reinterpret_cast<int*>(sbrk(0));
   return freeValue;
}
```

В процессе исполнения кода количество используемых кодом байтов памяти RAM постоянно меняется. Поэтому важно не допустить, чтобы код попытался использовать больший объем памяти RAM, чем доступный. В процессе исполнения скетча занятые объемы стека и кучи будут увеличиваться по направлению друг к другу. Если в какой-либо момент эти две области памяти столкнутся друг с другом, то у нас больше не будет свободной памяти RAM. При интенсивном использовании кучи она может *фрагментироваться*, т. е. область между начальным и конечным адресами кучи будет содержать участки свободной памяти. Рассмотренные в этом решении методы не учитывают такие прогалины в куче, поэтому объем свободной памяти RAM может быть чуть больше, чем они сообщают.

Далее приводится список основных процессов, потребляющих память RAM:

- при инициализации констант или определении препроцессорного макроса:

```
#define ERROR_MESSAGE "an error has occurred"
```

- при объявлении глобальных переменных:

```
char myMessage[] = "Hello World";
```

- при вызове функций:

```
void myFunction(int value)
{
   int result;
   result = value * 2;
   return result;
}
```

  Рекурсивные функции (функции, которые вызывают сами себя) с глубоким уровнем вложенности и/или большим количеством локальных переменных могут занимать очень большой объем памяти стека;

- при динамическом выделении памяти:

```
String stringOne = "Arduino String";
```

Класс `String` языка Arduino сохраняет строки в динамической памяти. Это можно увидеть, добавив следующую строку кода в самом начале кода скетча решения:

```
String s = "\n";
```

А непосредственно перед оператором задержки `delay()` в главном цикле `loop()` надо добавить следующие две строки кода:

```
s = s + "Hello I am Arduino\n";
Serial.println(s);              // Выводим в монитор порта значение строки
```

Вы увидите, что значение свободной памяти будет уменьшаться при увеличении размера строки на каждой итерации цикла. Если исполнять скетч в течение достаточно длительного времени, вся свободная память RAM будет использована, поэтом не пытайтесь бесконечно увеличивать размер строк в каком бы то ни было приложении, кроме тестового.

Код, который создает постоянно увеличивающееся значение, гарантированно вызовет нехватку памяти. Поэтому следует быть внимательным, чтобы не создать код, который в процессе исполнения динамически создает разные числа из переменных на основе какого-то параметра, поскольку в таком случае будет трудно проверить, что такой код не вызовет превышения возможностей памяти платы, на которой он исполняется.

Константы и глобальные переменные также часто объявляются в библиотеках, поэтому разработчик может не знать о них, тогда как они все же будут использовать память RAM. Например, библиотека Serial имеет глобальный массив размером в 128 байтов, который используется для хранения данных, поступающих по после-

довательному каналу связи. Только сам этот массив потребляет одну восьмую часть всей памяти RAM, доступной на старых микроконтроллерах AVR ATmega168.

Дополнительная информация

Более подробно об использовании памяти микроконтроллеров AVR рассказано на этой веб-странице: **https://oreil.ly/IgnwF**.

17.3. Использование программной флеш-памяти для записи и чтения числовых значений

ЗАДАЧА

Разрабатываемый скетч содержит большой объем постоянных числовых данных, для хранения которых не хочется использовать память RAM.

РЕШЕНИЕ

Числовые значения можно хранить в программной флеш-памяти микроконтроллера, используемой для хранения скетчей Arduino.

Скетч соответствующего решения приводится в листинге 17.3. Скетч вносит поправку на нелинейную чувствительность человеческого зрения в скорость изменения яркости светодиода. Для этого в скетче используется 256 числовых значений, которые скетч сохраняет не в памяти RAM для данных, а в программной флеш-памяти.

Этот скетч основан на скетче решения из *разд. 7.2*. Информация по подключению светодиодов и обеспечению их питания от внешнего источника питания приводится в *главе 7*. В результате исполнения этого скетча яркость светодиода, подключенного к контакту 5 платы Arduino, будет изменяться плавно по сравнению с изменением яркости светодиода, подключенного к контакту 3.

Листинг 17.3. Скетч для корректировки скорости изменения яркости светодиода

```
/*
* Скетч ProgmemCurve
* Преобразовывает линейное изменение выходного сигнала
  в экспоненциальное, используя таблицу значений, хранящуюся в программной памяти
*/

#include <avr/pgmspace.h> // Требуется для оператора PROGMEM

// Таблица экспоненциальных значений
// Созданная для значений i
// от 0 до 255 -> x=round( pow( 2.0, i/32.0) - 1)
```

```
const byte table[]PROGMEM =
{
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
   3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5,
   5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7,
   7, 7, 7, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 10, 10, 10,
   10, 11, 11, 11, 11, 12, 12, 12, 12, 13, 13, 13, 14, 14, 14, 15,
   15, 15, 16, 16, 16, 17, 17, 18, 18, 18, 19, 19, 20, 20, 21, 21,
   22, 22, 23, 23, 24, 24, 25, 25, 26, 26, 27, 28, 28, 29, 30, 30,
   31, 32, 32, 33, 34, 35, 35, 36, 37, 38, 39, 40, 40, 41, 42, 43,
   44, 45, 46, 47, 48, 49, 51, 52, 53, 54, 55, 56, 58, 59, 60, 62,
   63, 64, 66, 67, 69, 70, 72, 73, 75, 77, 78, 80, 82, 84, 86, 88,
   90, 91, 94, 96, 98, 100, 102, 104, 107, 109, 111, 114, 116, 119, 122, 124,
   127, 130, 133, 136, 139, 142, 145, 148, 151, 155, 158, 161, 165, 169, 172, 176,
   180, 184, 188, 192, 196, 201, 205, 210, 214, 219, 224, 229, 234, 239, 244, 250
};

const int rawLedPin = 3;       // Скорость изменения этого светодиода
                               // управляется неоткорректированными значениями
const int adjustedLedPin = 5; // Скорость изменения этого светодиода
                               // управляется откорректированными значения из таблицы

int brightness = 0;
int increment = 1;

void setup()
{
   // Для контактов, на которые выходной сигнал подается
   // функцией analogWrite(), не требуется явно задавать
   // выходной режим работы
}

void loop()
{
   if (brightness > 254)
   {
      increment = -1; // После достижения значения 255 начинаем обратный счет
   }
   else if (brightness < 1)
   {
      increment = 1; // После достижения значения 0 начинаем прямой счет
   }
   brightness = brightness + increment; // Увеличиваем яркость
              // (или уменьшаем, если значение increment = -1)
```

```
  // Записываем значение яркости на контакты управления светодиодами
  analogWrite(rawLedPin, brightness); // Неоткорректированное значение
  // Откорректированное значение
  int adjustedBrightness = pgm_read_byte(&table[brightness]);
  analogWrite(adjustedLedPin, adjustedBrightness);

  delay(10); // 10 мс задержка в каждой итерации цикла означает
             // длительность в 2,55 секунды периода уменьшения
             // или увеличения яркости светодиодов
}
```

Обсуждение работы решения и возможных проблем

Когда для вычисления диапазона регулярно повторяющихся значений используется сложное выражение, зачастую удобнее вычислить эти значения предварительно и включить их в код как таблицу значений (обычно в виде массива). Это позволит сэкономить время, требуемое для постоянного вычисления требуемых значений в ходе исполнения кода. Недостаток этого метода состоит в необходимости использования памяти RAM для хранения всех этих значений. Но поскольку микроконтроллеры плат Arduino имеют ограниченных объем памяти RAM, такие постоянные значения лучше сохранить в программной памяти, объем которой намного больше. Это особенно полезно для скетчей, использующих большие массивы чисел.

Для этого в начале скетча определяем таблицу в виде массива следующим образом:

```
const byte table[]PROGMEM =
{
   0, . . .
```

Оператор `PROGMEM` дает указание компилятору сохранить значения массива не в памяти RAM, а в программной флеш-памяти. Если из скетча для платы Arduino удалить оператор `PROGMEM`, использование памяти RAM глобальными переменными взлетит от 13 байтов до 269 байтов (и такой скетч работать не будет, поскольку оператор `pgm_read_byte` не сможет правильно функционировать без оператора `PROGMEM` в объявлении массива). Остальная часть объявления массива такая же, как и при объявлении обычного массива (см. *главу 2*).

Низкоуровневые определения, требуемые для использования оператора `PROGMEM`, содержатся в заголовочном файле pgmspace.h, который подключается к скетчу следующим образом:

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

Хотя путь к этому заголовочному файлу содержит элемент `avr`, его можно также задействовать и для плат с микроконтроллерами с 32-разрядной архитектурой, поскольку элемент `avr` используется только для обратной совместимости с платами с микроконтроллерами AVR. Реализация решения для плат с микроконтроллерами с 32-разрядной архитектурой будет немного проще, и действительно, на платах с микроконтроллерами ARM (SAM, SAMD) макрос `PROGMEM` объявляется пустым. Это объясняется тем, что компилятор ARM обычно сохраняет структуры данных,

объявленных как константы (const), в программной флеш-памяти. На платах с микроконтроллером ARM местонахождение таких данных можно проверить, добавив в главный цикл loop() следующий код:

```
Serial.begin(9600);
while(!Serial); // Для плат Leonardo и других 32-разрядных плат
Serial.print("Address of table: 0x"); // Адрес таблицы:
Serial.println((int)&table, HEX);
```

Если отображается значение в диапазоне от 0x0000 до 0x3FFFF, то данные хранятся в программной памяти. Если же из объявления таблицы table убрать ключевое слово const, то таблица будет храниться по намного более высокому адресу (0x2000000 или выше).

Чтобы внести поправку в скорость изменения яркости светодиода, чтобы она выглядела постоянной для человеческого глаза, в код управления светодиодом из скетча решения *разд. 7.2* (см. листинг 7.2) добавляются следующие две строки кода:

```
int adjustedBrightness = pgm_read_byte(&table[brightness]);
analogWrite(adjustedLedPin, adjustedBrightness);
```

Переменной adjustedBrightness присваивается значение, считываемое из программной памяти. Выражение pgm_read_byte(&table[brightness]); возвращает адрес входа в массив table по указателю, предоставляемому значением переменной brightness.

Дополнительная информация

Публикация Memories of an Arduino (**https://cdn-learn.adafruit.com/downloads/pdf/memories-of-an-arduino.pdf**) компании Adafruit содержит много полезных советов и методов по работе с памятью плат Arduino.

В *разд. 17.4* рассматривается метод, обеспечивающий хранение строковых данных в программной флеш-памяти, впервые примененный в версии 1.0 среды Arduino IDE.

17.4. Использование программной флеш-памяти для записи и чтения строковых значений

ЗАДАЧА

Разрабатываемый вами скетч содержит много постоянных строковых данных, которые потребляют слишком большой объем памяти RAM. Нужно сохранить эти постоянные строковые данные — например, пункты меню или отладочные операторы, — не в памяти RAM, а в программной флеш-памяти.

РЕШЕНИЕ

Эта задача решается скетчем, код которого приводится в листинге 17.4. Скетч сохраняет строку в программной памяти и выводит ее значение в окно монитора пор-

та, используя выражение `F("текст")`. Метод для определения и вывода на экран объема свободной памяти RAM рассматривается в *разд. 17.2*. Этот скетч совмещает подходы для использования с платами как на микроконтроллере ARM, так и на микроконтроллере AVR. Но для плат с каким-либо другим микроконтроллером этот скетч, скорее всего, не скомпилируется должным образом.

Листинг 17.4. Скетч для сохранения строковых данных в программной флеш-памяти

```
/*
* Скетч Write strings using Program memory (Flash)
*/

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

void loop()
{
   Serial.println(memoryFree()); // Выводим в окно монитора порта объем
                                 // свободной памяти RAM
   Serial.println(F("Arduino")); // Выводим в монитор порта значение строки
   delay(1000);
}

#ifdef __arm__
   // Переменная, созданная процессом сборки при компилировании скетча
   extern "C" char *sbrk(int incr); // Вызывается с аргументом 0,
          // чтобы получить начальный адрес свободной памяти RAM
#else
   extern int *__brkval; // Указатель на адрес последней выделенной
                         // ячейки кучи (или 0)
#endif

// Функция, возвращающая объем свободной памяти RAM
int memoryFree()
{
int freeValue; // Это будет самый последний объект,
               // для которого выделена память в куче
#ifdef __arm__
   freeValue = &freeValue - reinterpret_cast<int*>(sbrk(0));
#else
   if((int)__brkval == 0) // Куча пустая. Используем адрес начала кучи
   {
      freeValue = ((int)&freeValue) - ((int)__malloc_heap_start);
   }
```

```
    else // Куча непустая; используем последний адрес в куче
    {
        freeValue = ((int)&freeValue) - ((int)__brkval);
    }
    #endif
    return freeValue;
}
```

Обсуждение работы решения и возможных проблем

Строки — это особенно ресурсоемкие элементы в плане расходования памяти RAM. Для хранения одного символа требуется один байт памяти, так что при использовании в скетче большого количества слов очень легко занять ими большой объем памяти RAM. Заключение текста в выражение `F("текст")` сохраняет его в намного большей программной флеш-памяти вместо памяти RAM.

Если строку сохранить, не заключая ее в это выражение, то можно будет увидеть, что объем свободной памяти окажется меньшим, чем при использовании выражения (по крайней мере, в платах с микроконтроллером AVR). Для плат с микроконтроллером ARM — в зависимости от того, как компилятор ARM оптимизирует код, — строки могут помещаться в программную флеш-память без заключения их в выражение `F("текст")`.

Дополнительная информация

Использование программной флеш-памяти для хранения строк веб-страницы рассматривается в решении из *разд. 15.13*.

17.5. Использование вместо целых чисел ключевых слов *#define* и *const*

ЗАДАЧА

Требуется минимизировать использование памяти RAM, дав указание компилятору, что значение является константой и его можно оптимизировать.

РЕШЕНИЕ

Эта задача решается использованием ключевого слова `const` для объявления постоянным числовых значений в скетче. Например, вместо следующего выражения:

```
int ledPin = 2;
```

нужно использовать это:

```
const int ledPin = 2;
```

Обсуждение работы решения и возможных проблем

Часто возникает потребность использовать в разных областях кода постоянные числовые значения. Использование в таких случаях собственно чисел крайне не рекомендуется. Если в дальнейшем нужно будет изменить какое-либо такое значение, придется найти все места его расположения в коде, что при достаточно большом объеме кода и достаточно большом количестве значений окажется задачей, которую будет трудно решить, не пропустив одно или даже несколько из этих значений. Поэтому лучше всего использовать именованные ссылки.

Постоянные числовые значения (константы) можно объявлять тремя разными способами:

- как переменную, значение которой сохраняется в памяти RAM:

```
int ledPin = 2; // переменная расходует память RAM
```

- как константу, значение которой сохраняется в программной флеш-памяти:

```
const int ledPin = 2; // константы не используют память RAM
```

- используя директиву `#define`, которая дает указание препроцессору заменить идентификатор (в нашем случае `ledPin`) числовым значением (в нашем случае `LED_BUILTIN`):

```
#define ledPin LED_BUILTIN
```

Идентификатор, объявленный любым из этих трех способов, используется в коде абсолютно одинаково:

```
pinMode(ledPin, OUTPUT);
```

Хотя первые два выражения выглядят похожими друг на друга, ключевое слово `const` во втором из них сообщает компилятору, что идентификатор `ledPin` не является обычной переменной. В отличие от переменной, объявленной просто с использованием ключевого слова `int`, в результате добавления ключевого слова `const` для значения не резервируется место в памяти RAM, поскольку это ключевое слово гарантирует, что значение не будет меняться. Для выражения с идентификатором, объявленным с использованием ключевого слова `const`, компилятор создаст точно такой же исполняемый код, как будто бы было использовано просто числовое значение:

```
pinMode(2, OUTPUT);
```

Впрочем, если идентификатор `ledPin`, объявленный без использования ключевого слова `const`, используется в скетче таким образом, как будто бы он был константой (т. е. его значение никогда не модифицируется), компилятор, скорее всего, заметит это обстоятельство и оптимизирует этот идентификатор в любом случае, создавав исполняемый код с использованием постоянного числового значения.

В коде скетчей более старых версий Arduino IDE иногда можно видеть определение констант на основе директивы `#define`, однако лучше использовать вместо этой директивы ключевое слово `const`. Это объясняется тем, что значение `const` имеет тип

данных, что позволяет компилятору проверить, используется ли оно должным образом для этого типа, и выдать предупреждение, если нет. Компилятор также следует правилам языка C касательно области видимости значений `const`. А значение, объявленное посредством директивы `#define`, будет видимым во всем скетче, что может не соответствовать вашим намерениям. Еще одно достоинство использования ключевого слова `const` состоит в том, что его выражение имеет знакомый синтаксис, тогда как выражение на основе директивы `#define` не использует знак равенства, а также не завершается точкой с запятой. Однако одним из достоинств директивы `#define` является возможность использования ее для условного компилирования кода, когда компилятор использует или игнорирует фрагменты исходного кода в зависимости от значения идентификатора одной или нескольких директив `#define`. Пример такого условного компилирования приводится в *разд. 17.6*.

Дополнительная информация

Более подробная информация по работе с препроцессором приведена в *разд. 17.0*.

17.6. Условное компилирование

ЗАДАЧА

Требуется выполнить компиляцию того или другого фрагмента кода, в зависимости от определенного условия. Например, скомпилировать одну версию кода для отладки, а другую — для исполнения в реальных условиях, или скомпилировать разные фрагменты кода для разных плат.

РЕШЕНИЕ

Эту задачу можно решить, используя условные операторы, на основании которых препроцессор решает, какие фрагменты кода компилировать, а какие нет.

Пример скетча решения приводится в листинге 17.5. Скетч основан на скетче решения из *разд. 5.6* и выводит в окно монитора порта отладочные сообщения при условии наличия определения с помощью директивы `#define` идентификатора `DEBUG`.

Листинг 17.5. Скетч с условием компилирования

```
/*
Скетч Pot_Debug
Мигает светодиодом с частотой, заданной вращением потенциометра
Выводит значения в окно монитора порта, если определен
идентификатор DEBUG
*/

const int potPin = 0;   // Контакт для подключения потенциометра
const int ledPin = 13;  // Контакт для подключения светодиода
```

```
int val = 0; // Переменная для хранения значения, полученного
             // с датчика (потенциометра)

#define DEBUG

void setup()
{
   Serial.begin(9600);
   pinMode(ledPin, OUTPUT); // Задаем выходной режим работы контакта,
                            // к которому подключен встроенный светодиод
}

void loop()
{
   val = analogRead(potPin);       // Считываем напряжение на контакте потенциометра
   digitalWrite(ledPin, HIGH);     // Включаем встроенный светодиод
   delay(val);                     // Частота мигания, заданная
                                   // значением сигнала с потенциометра
   digitalWrite(ledPin, LOW);      // Выключаем светодиод
   delay(val);                     // Включает светодиод на такое же время

   #if defined DEBUG
      Serial.println(val);
   #endif
}
```

Обсуждение работы решения и возможных проблем

В этом скетче препроцессор в начале процесса компиляции решает, какие фрагменты кода компилировать. В частности, в скетче выполняется проверка, был ли объявлен идентификатор DEBUG, и в случае положительного результата проверки в исполняемый код включается часть скетча, которая отвечает за вывод отладочных сообщений в окно монитора порта. Выражения, которые начинаются с символа #, обрабатываются препроцессором, прежде чем выполняется компилирование (тема препроцессора рассматривается более подробно в *разд. 17.0*).

Условное компилирование можно также выполнять в зависимости от типа микроконтроллера платы. Например, в листинге 17.6 приводится скетч, который при компилировании для плат Arduino Mega создает другой код, который считывает дополнительные контакты этой платы.

Листинг 17.6. Скетч с условным компилированием в зависимости от платы

```
/*
* Скетч ConditionalCompile
* Компилируется по-разному для разных микроконтроллеров,
  с учетом условной директивы define
*/
```

```
int numberOfSensors;
int val = 0; // Переменная для хранения значения, полученного
             // с датчика (потенциометра)

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

   #if defined(__AVR_ATmega2560__) // Определяется, если
               // в среде Arduino IDE выбрана плата Mega
      numberOfSensors = 16; // Количество входных аналоговых контактов
                            // на плате Mega
      #pragma message ( "Using 16 sensors" ) // Используем 16 датчиков
   #else // Если плата не Mega, предполагается стандартная плата
      numberOfSensors = 6; // Количество входных аналоговых контактов
                           // на стандартной плате Arduino
      #pragma message ( "Using 6 sensors" ) // Используем 6 датчиков
   #endif

   Serial.print("The number of sensors is "); // Количество датчиков:
   Serial.println(numberOfSensors);
}

void loop()
{
   for(int sensor = 0; sensor < numberOfSensors; sensor++)
   {
      val = analogRead(sensor); // Считываем значение сигнала датчика
      Serial.print(sensor); Serial.print(": ");
      Serial.println(val);      // Отображаем полученное значение
   }
   Serial.println();
   delay(1000); // Секундная пауза перед между считываниями
}
```

При компилировании скетча для платы Mega директива `#pragma` выводит в панели сообщений среды Arduino IDE следующее сообщение:

```
C:\Sketches\conditional.ino:15:40: note: #pragma message: Using 16 sensors
         #pragma message ( "Using 16 sensors" )
```

Дополнительная информация

На веб-сайте The C Preprocessor (**https://oreil.ly/kh_XU**) приводится обширная и подробная информация о препроцессоре языка C.

ГЛАВА 18

Работа с аппаратными средствами микроконтроллера платы

18.0. Введение

Платформа Arduino позволяет упростить разработку программ, предоставляя пользователям простые в работе функции, которые скрывают сложные низкоуровневые операции аппаратных средств платформы. Но для некоторых приложений требуется прямой доступ к аппаратному обеспечению, минуя функции высшего уровня. Это может быть полезно в тех случаях, когда нужно получить более высокую эффективность или когда это единственный способ решить поставленную задачу. В этой главе мы рассмотрим, как получить доступ к аппаратным функциям, которые не полностью доступны в среде программирования Arduino, и использовать их.

Прежде чем приступать к углубленному рассмотрению аппаратных функций платформы Arduino, необходимо получить представление о четырех ключевых составляющих ее аппаратного обеспечения: регистрах, прерываниях, таймерах и цифровых и аналоговых контактах.

- *Регистры* — это ячейки памяти микроконтроллера, с помощью которых можно управлять его работой. Регистры часто используются для настройки режима работы других аппаратных средств микроконтроллера.
- *Прерываниями* называются сигналы, создаваемые микроконтроллером обычно в ответ на внешнее событие, и позволяющие скетчам Arduino немедленно реагировать на такое событие.
- *Таймеры* создают сигнал после предопределенной задержки или генерируют сигналы, повторяющиеся с заданным периодом. Подобно прерываниям, скетч может реагировать в ответ на событие сигнала таймера.
- Мы уже узнали из предыдущих глав, как работать с *аналоговыми* и *цифровыми контактами*, но в решениях этой главы демонстрируется, как использовать эти аппаратные элементы намного более эффективно.

Модифицирование значений регистров может изменить поведение некоторых функций Arduino (например, функции `millis()`). При использовании рассматриваемых в этой главе низкоуровневых возможностей платформы Arduino следует быть внимательным, соблюдать осторожность и выполнять предварительное тестирование, чтобы обеспечить правильное функционирования кода.

Рассмотрим все эти четыре основные категории аппаратных средств Arduino несколько более подробно.

Регистры

Регистры можно рассматривать как переменные, указывающие на ячейки памяти микроконтроллера. Регистры используются микроконтроллером для настройки аппаратных функций или для хранения результатов аппаратных вычислений. Значения регистров можно записывать и считывать с помощью скетча. Модификация значения регистров может изменять режим работы или состояние аппаратных средств микроконтроллера (например, выходное состояние контакта). Некоторые регистры представляют числовое значение (например, предел счета таймера). Регистры могут задавать или сообщать о статусе аппаратных средств — например, о состоянии контакта или наличии прерывания. Обращение к регистрам в коде осуществляется с использованием названий регистров, которые содержатся в документации на микроконтроллер. Присвоение регистру неправильного значения обычно влечет за собой неправильную работу скетча, поэтому внимательно изучите работу с ними в документации, чтобы использовать их должным образом.

Прерывания

Прерывания представляют собой сигналы, которые позволяют микроконтроллеру остановить обычный ход исполнения скетча и уделить внимание обработке задачи, требующей немедленного внимания, а затем возвратиться обратно к выполнению прерванного процесса. Базовое программное обеспечение Arduino использует прерывания для обработки входящих данных последовательного порта, для отсчета времени функциями `delay()` и `millis()` и для активирования функции `attachInterrupt()`. Прерывания при возникновении события также используются библиотеками Wire и Servo — чтобы коду не требовалось постоянно проверять, произошло ли событие. Такие постоянные проверки, которые называются *опросом*, могут усложнить логику разрабатываемого скетча. Прерывания могут быть надежным способом обнаружения очень быстротечных сигналов. В *разд. 18.2* показано, как с помощью прерываний определить, не изменилось ли состояние цифрового контакта.

В процессе обработки одного прерывания может возникнуть другое прерывание или даже несколько других прерываний. Например, при нажатии двух кнопок, каждая из которых создает отдельное прерывание. В таком случае, прежде чем приступать к обработке нового прерывания, необходимо завершить обработку текущего прерывания. Код обработчика прерывания должен исполняться в течение как можно более короткого времени, поскольку, если исполнение занимает слишком много времени, это может вызвать задержку с обработкой других ожидающих прерываний или даже пропуск события прерывания.

Микроконтроллер платы Arduino может обрабатывать одновременно только одно прерывание. В процессе обработки одного прерывания все поступающие новые прерывания ставятся в очередь ожидания обработки. Код обработчика прерывания должен исполняться в течение как можно более короткого времени, чтобы не вызывать чрезмерных задержек с обработкой других ожидающих прерываний. Обработчик прерывания, занимающий слишком много времени, может вызвать пропуск обработки других прерываний. Следует избегать использования прерываний для задач, исполнение которых занимает сравнительно длительное время, — напри-

мер, мигания светодиодом или даже вывода текста через последовательный интерфейс. Если необходимо выполнить такую задачу, то в обработчике прерывания можно установить глобальный флаг и использовать этот флаг, чтобы дать указание коду выполнять эту задачу в главном цикле.

Таймеры

Плата Arduino Uno (и совместимые платы с микроконтроллером ATmega328) оснащена тремя аппаратными таймерами для управления задачами, требующими ведения отсчета времени (платы Mega оснащены шестью таймерами — см. далее). Эти таймеры используются несколькими функциями Arduino:

- таймер Timer0 — функциями `millis()` и `delay()`, а также функцией `analogWrite()` при записи на контакты 5 и 6;
- таймер Timer1 — функцией `analogWrite()` при записи на контакты 9 и 10, а также библиотекой Servo для управления сервоприводами;
- таймер Timer2 — функцией `analogWrite()` при записи на контакты 3 и 11.

Библиотека Servo использует тот же таймер, что и функция `analogWrite()` при записи на контакты 9 и 10, поэтому при работе с библиотекой Servo функцию `analogWrite()` нельзя задействовать для записи на контакты 9 и 10.

Плата Mega оснащена тремя дополнительными 16-разрядными таймерами и используется функцией `analogWrite()` для записи на следующие контакты:

- таймер Timer0 — на контакты 4 и 13;
- таймер Timer1 — на контакты 11 и 12;
- таймер Timer2 — на контакты 9 и 10;
- таймер Timer3 — на контакты 2, 3 и 5;
- таймер Timer4 — на контакты 6, 7 и 8;
- таймер Timer5 — на контакты 45 и 46.

Таймерная система, используемая в платах с микроконтроллером megaAVR (например, Arduino WiFi Rev2 или Arduino Nano Every), отличается от таймерной системы плат с микроконтроллером ATmega (например, Arduino Uno или Mega). Реализация таймеров в микроконтроллерах megaAVR объясняется в документе по их применению от компании Microchip (**https://oreil.ly/7H4WW**). В платах Arduino с микроконтроллером ARM используется совершенно иной подход к работе с таймерами, основанный на двух средствах: Timer Counter (TC) и Timer Counter for Control Applications (TCC). Оба эти средства подробно рассматриваются в следующих документах по их применению: Timer Counter (TC) Driver (**https://oreil.ly/FTDOT**) и Timer Counter for Control Applications (TCC) Driver (**https://oreil.ly/RTXwQ**).

Компания Adafruit предоставляет на странице GitHub (**https://oreil.ly/ccQXc**) свою библиотеку Adafruit_ZeroTimer, в состав которой входит оберточный код для модулей Timer Control 3, 4 и 5, и поддерживающую как платформу SAMD21 (ядро ARM Cortex-M0), так и платформу SAMD51 (ядро ARM Cortex-M4). Однако эта библиотека не предназначена для использования вне пределов компании Adafruit и не поддерживается никем, кроме самой компании.

Таймер — это счетчик, который считает импульсы, выдаваемые источником времени (timebase, временной базой). Аппаратная часть таймера состоит из 8-разрядных или 16-разрядных счетчиков, которые можно запрограммировать для определения режима счета, используемого таймером. Наиболее часто используется режим, в котором считаются импульсы временной базы платы Arduino, обычно с частотой 16 МГц, получаемые с кварцевого генератора колебаний. Период импульсов с частотой 16 МГц составляет 62,5 наносекунд, что слишком быстро для многих приложений с использованием времени, поэтому частота сигнала временной базы понижается с помощью предварительного делителя частоты. Например, разделив частоту сигнала временной базы на 8, мы увеличим длительность каждого импульса до половины микросекунды. Для приложений, которым и это слишком быстро, можно использовать другие значения предварительного делителя частоты (см. табл. 18.1).

Работа таймера управляется с помощью значений регистров, которые можно записывать и считывать, используя код Arduino. В частности, значения регистров задают частоту таймера (количество импульсов сигнала временной базы между каждым счетом) и режим счета (прямой, обратный, прямой и обратный или с использованием внешнего сигнала).

Далее приводится список регистров управления таймерами и их краткое описание (буква *n* обозначает номер таймера):

- **Timer/Counter Control Register A (TCCR*n*A)** — регистр A управления таймером/счетчиком. Определяет режим работы;
- **Timer/Counter Control Register B (TCCR*n*B)** — регистр B управления таймером/счетчиком. Определяет значение предварительного делителя частоты;
- **Timer/Counter Register (TCNT*n*)** — регистр таймера/счетчика. Содержит счет таймера;
- **Output Compare Register A (OCR*n*A)** — регистр A сравнения вывода. По этому счету можно активировать прерывания;
- **Output Compare Register B (OCR*n*B)** — регистр B сравнения вывода. По этому счету можно активировать прерывания;
- **Timer/Counter Interrupt Mask Register (TIMSK*n*)** — регистр маски прерываний таймера/счетчика. Задает условия для активирования прерывания;
- **Timer/Counter Interrupt Flag Register (TIFR*n*)** — регистр флага прерывания таймера/счетчика. Указывает, произошло ли прерывание.

В табл. 18.1 приводятся значения битов, применяемых для задания точности таймера. Функции регистров подробно рассматриваются в решениях, в которых они используются.

Все таймеры инициализируются коэффициентом предварительного делителя, равным 64.

Точность в наносекундах равна периоду ЦПУ (время одного цикла ЦПУ), умноженному на коэффициент предварительного делителя.

Таблица 18.1. *Значения предварительного делителя частоты (сигнал тактирования частотой 16 МГц)*

Коэффициент предварительного делителя частоты	CSx2, CSx1, CSx0	Точность	Время для переполнения	
			8-разрядного таймера	16-разрядного таймера
1	B001	62,5 нс	16 мкс	4,096 мс
8	B010	500 нс	128 мкс	32,768 мс
64	B011	4 мкс	1,024 мкс	262,144 мс
256	B100	16 мкс	4,096 мкс	1048,576 мс
1024	B101	64 мкс	16,384 кс	4194,304 мс
	B110	Внешний сигнал тактирования, отрицательный перепад		
	B111	Внешний сигнал тактирования, положительный перепад		

Аналоговые и цифровые контакты

В *главе 5* мы рассмотрели стандартные функции Arduino для записи на цифровые и аналоговые контакты и чтения с них. В этой же главе мы рассмотрим способы более быстрого управления контактами, чем дают возможность стандартные функции Arduino, а также модифицируем аналоговые функции, чтобы улучшить производительность.

Некоторый код, приведенный в этой главе, может оказаться более сложным для понимания, чем код других решений этой книги, поскольку он выходит за пределы синтаксиса языка Arduino и впрямую управляет базовым аппаратным обеспечением. Код в решениях этой главы работает непосредственно с регистрами микроконтроллера, используя их краткие названия и манипулируя их битами с помощью операций битовых сдвигов и маскирования. Польза, которую приносит эта сложность, состоит в улучшении функциональности и рабочих характеристик.

Дополнительная информация

- Обзор аппаратных ресурсов приведен здесь: **https://oreil.ly/lqHdZ**.
- Библиотека Timer1 описана здесь: **https://oreil.ly/r8kPe**, а Timer3 — здесь: **https://oreil.ly/Q0hIS**.
- Учебное пособие по таймерам и ШИМ представлено здесь: **https://oreil.ly/oiTWN**.
- Справочные листки по линейке микроконтроллеров ATmega328 приведены здесь: **https://oreil.ly/vGUYT**.
- Документ компании Microchip по настройке и использованию таймеров приведен здесь: **https://oreil.ly/AEs0V**.

- Статью в английской Wikipedia по прерываниям вы найдете здесь: **https://oreil.ly/BIVfo**.

18.1. Запись данных в память EEPROM

ЗАДАЧА

Требуется сохранить некоторые данные таким образом, чтобы они были доступными после выключения питания.

РЕШЕНИЕ

Эта задача решается записью данных в память EEPROM с помощью библиотеки EEPROM. Соответствующий скетч приводится в листинге 18.1 (он основан на скетче BlinkWithoutDelay, упоминание о котором вы найдете в *разд. «Обсуждение работы решения и возможных проблем» разд. 12.2*). Скетч мигает светодиодом на основе значений, считываемых из памяти EEPROM, а также позволяет записывать в нее новые значения через монитор порта.

Листинг 18.1. Скетч для записи данных в памяти EEPROM и чтения их оттуда

```
/*
*  Скетч EEPROM основан на скетче BlinkWithoutDelay
* Сохраняет значения частоты мигания в памяти EEPROM
*/

#include <EEPROM.h>

// Значения этих переменных сохраняются в памяти EEPROM
const byte EEPROM_ID = 0x99; // Используется для определения наличия
                             // действительных данных в памяти EEPROM
byte ledPin = LED_BUILTIN;   // Контакт встроенного светодиода
int interval = 1000;         // Период мигания (в миллисекундах)

// Переменные, которые не требуется сохранять
int ledState = LOW;      // Переменная для установки состояния светодиода
long previousMillis = 0; // Переменная для хранения последнего времени
                         // обновления светодиода

// Константы для адресов ячеек памяти EEPROM
const int ID_ADDR = 0;       // Адрес ячейки памяти EEPROM для хранения ID-номера
const int PIN_ADDR = 1;      // Адрес ячейки памяти EEPROM
                             // для хранения номера контакта
const int INTERVAL_ADDR = 2; // Адрес ячейки памяти EEPROM
                             // для хранения значения периода мигания

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

```
   byte id = EEPROM.read(ID_ADDR); // Считываем первый байт из памяти EEPROM
   if( id == EEPROM_ID)
   {
      // Переходим сюда, если считанное значение id совпадает
      // со значением, сохраненным при записи в память EEPROM
      Serial.println("Using data from EEPROM"); // Используем данные из EEPROM
      ledPin = EEPROM.read(PIN_ADDR);
      byte hiByte = EEPROM.read(INTERVAL_ADDR);
      byte lowByte = EEPROM.read(INTERVAL_ADDR+1);
      interval = word(hiByte, lowByte); // Функция word()
                                        // рассматривается в разд. 3.15
   }
   else
   {
      // Переходим сюда, если ID не обнаружен, поэтому записываем
      // данные по умолчанию
      Serial.println("Writing default data to EEPROM");
                  // Записываем в EEPROM данные по умолчанию
      EEPROM.write(ID_ADDR,EEPROM_ID); // Записываем ID для обозначения
                                       // действительных данных
      EEPROM.write(PIN_ADDR, ledPin);  // Сохраняем номер контакта в EEPROM
      byte hiByte = highByte(interval);
      byte loByte = lowByte(interval);
      EEPROM.write(INTERVAL_ADDR, hiByte);
      EEPROM.write(INTERVAL_ADDR+1, loByte);
   }
   Serial.print("Setting pin to "); // Светодиод подключен к контакту
   Serial.println(ledPin,DEC);
   Serial.print("Setting interval to "); // Задаем период мигания, равный
   Serial.println(interval);

   pinMode(ledPin, OUTPUT);
}

void loop()
{
   // Это такой же код, что и в скетче примера BlinkWithoutDelay
   if (millis() - previousMillis > interval)
   {
      previousMillis = millis(); // Сохраняем последнее время изменения
                                 // состояния светодиода
      // Если светодиод выключен, включаем его, и наоборот
      if (ledState == LOW)
         ledState = HIGH;
      else
         ledState = LOW;
      digitalWrite(ledPin, ledState); // Задаем состояние светодиода,
                                      // используя значение переменной ledState
   }
```

```
   processSerial();
}

// Функция для установки периода мигания и номера контакта светодиода
// через монитор порта
// Значение, за которым следует буква i, задает период мигания
int value = 0;
void processSerial()
{
   if( Serial.available())
   {
      char ch = Serial.read();
      if(ch >= '0' && ch <= '9') // Проверяем, что принят
                                 // символ ASCII от 0 до 9
      {
         value = (value * 10) + (ch - '0'); // Да, накапливаем значения
      }
      else if (ch == 'i') // Проверяем, задает ли это значение период мигания
      {
         interval = value;
         Serial.print("Setting interval to "); // Задаем период мигания,
                                               // равный
         Serial.println(interval);
         byte hiByte = highByte(interval);
         byte loByte = lowByte(interval);
         EEPROM.write(INTERVAL_ADDR, hiByte);
         EEPROM.write(INTERVAL_ADDR+1, loByte);
         value = 0; // Обнуляем, чтобы быть готовым к приему следующей
                    // последовательности цифр
      }
      else if (ch == 'p') // Проверяем, задает ли это значение номер контакта
      {
         ledPin = value;
         Serial.print("Setting pin to "); // Светодиод подключен к контакту
         Serial.println(ledPin,DEC);
         pinMode(ledPin, OUTPUT);
         EEPROM.write(PIN_ADDR, ledPin); // Сохраняем номер контакта в EEPROM
         value = 0; // Обнуляем, чтобы быть готовым к приему следующей
                    // последовательности цифр
      }
   }
}
```

Запустите программу монитора порта. В начале самого первого исполнения скетча в окно монитора порта выводится сообщение о том, используются ли значения, сохраненные ранее в памяти EEPROM, или же значения по умолчанию.

Значения можно изменить, введя в строку передачи монитора порта требуемое число, а за ним букву, указывающую назначение этого числа. Значение с буквой `i` задает новый период мигания светодиода, а с буквой `p` — номер контакта для подключения светодиода.

Обсуждение работы решения и возможных проблем

Микроконтроллер платы Arduino оснащен встроенной энергонезависимой памятью EEPROM, которая удерживает записанные в нее данные даже после выключения питания. Объем памяти EEPROM на плате Arduino Uno составляет 1024 байта, а на плате Mega — 4096 байта (4 Кбайт). Платы Arduino Uno WiFi R2 и Nano Every имеют всего лишь 256 байтов памяти EEPROM. Большинство плат с микроконтроллером ARM вообще не имеют памяти EEPROM.

Для записи и чтения данных в памяти EEPROM скетч использует библиотеку EEPROM. Подключение этой библиотеки к скетчу делает доступным объект `EEPROM`, который используется для работы с этой памятью, предоставляя методы для ее чтения, записи и обнуления. Метод `EEPROM.clear()` не используется в этом скетче, поскольку он обнуляет всю память EEPROM.

Для записи или чтения памяти EEPROM соответствующему методу библиотеки EEPROM необходимо указать адрес ячейки памяти, которую нужно записать или прочитать. Это означает, что разработчик должен самостоятельно отслеживать, в какую ячейку что записывается, чтобы потом можно было знать адрес, с которого считывать требуемые значения. Для записи используется метод `EEPROM.write(`*адрес, значение*`)`. Значение адреса может быть в диапазоне от 0 до 1023 (для платы Uno), а значение — один байт. Для чтения используется метод `EEPROM.read(`*адрес*`)`, который возвращает байт, содержащийся в ячейке памяти с указанным адресом.

Скетч решения сохраняет в памяти EEPROM три значения. Сначала сохраняется значение `ID`, которое используется только в функции `setup()`, чтобы определить, записывались ли уже в память EEPROM действительные данные или нет. Если сохраненное в памяти EEPROM значение `ID` совпадает с ожидаемым значением, это означает, что скетч уже исполнялся на этой плате и сохранил в ее памяти EEPROM и другие два значения. В таком случае эти два значения считываются из памяти EEPROM и используются в скетче. Если же значения не совпадают, то это означает, что скетч не исполнялся на этой плате (ибо в противном случае в ее EEPROM было бы записано значение `ID`). В таком случае в память EEPROM записываются значения по умолчанию, включая значение `ID`.

Скетч отслеживает последовательный порт и записывает поступающие через него значения в память EEPROM: значение `ID` — по адресу 0, номер контакта — по адресу 1, а два байта для длительности периода — по начальному адресу 2. Следующая строка кода записывает в память EEPROM номер контакта. Размер переменной `ledPin` составляет один байт, так что ее значение помещается в одну ячейку памяти EEPROM:

```
EEPROM.write(PIN_ADDR, ledPin); // Сохраняем номер контакта в EEPROM
```

Поскольку переменная периода мигания `interval` имеет тип `int`, для нее требуются два байта памяти:

```
byte hiByte = highByte(interval);
byte loByte = lowByte(interval);
EEPROM.write(INTERVAL_ADDR, hiByte);
EEPROM.write(INTERVAL_ADDR+1, loByte);
```

Код для записи значения периода мигания разделяет это значение на два байта, которые сохраняются по смежным адресам. Любые значения, записываемые после этого в память EEPROM, нужно записывать по адресу, следующему после этих двух байтов.

Значение `interval` считывается из памяти EEPROM по байтам, а затем восстанавливается в одно целое:

```
ledPin = EEPROM.read(PIN_ADDR);
byte hiByte = EEPROM.read(INTERVAL_ADDR);
byte lowByte = EEPROM.read(INTERVAL_ADDR+1);
interval = word(hiByte, lowByte);
```

Подробная информация по использованию функции `word()` для создания целочисленного значения типа `int` из двух байтов приводится в *главе 3*.

Для более сложной работы с памятью EEPROM рекомендуется создать карту, фиксирующую, что и по каким адресам сохраняется, чтобы не допустить попыток записи в одну ячейку памяти двух разных значений, а также чтобы избежать перезаписи многобайтными значениями другой информации.

Дополнительная информация

В *разд. 3.14* приводится более подробная информация по разбиению 16 и 32-разрядных значений на байты.

18.2. Автоматическое реагирование на изменение состояния контакта

ЗАДАЧА

Требуется выполнять определенное действие при изменении состояния цифрового контакта, но при этом без постоянной проверки состояния этого контакта.

РЕШЕНИЕ

Скетч из листинга 18.2 отслеживает импульсы на контакте 2 и сохраняет их длительность в массив. По заполнении массива (получению 32 импульсов) длительность каждого импульса выводится в окно монитора порта.

В случае использовании платы семейства Arduino MKR подключите светодиод к контакту 4 и модифицируйте код соответствующим образом.

Листинг 18.2. Скетч отслеживает импульсы на контакте 2 и сохраняет их длительность в массив

```
/*
* Скетч Interrupts
* Подключение датчика показано на рис. 10.1.
*/

const int pin = 2;              // Приемник ИК-сигнала подключен
                                // к контакту 2 платы Arduino
const int numberOfEntries = 32; // Этой константе можно присвоить
                                // любое удобное значение

volatile unsigned long microseconds;
volatile byte idx = 0;
volatile unsigned long results[numberOfEntries];

void setup()
{
   pinMode(pin, INPUT_PULLUP);
   Serial.begin(9600);
   // Используем прерывание контакта для обнаружения изменений
   attachInterrupt(digitalPinToInterrupt(pin), analyze, CHANGE);
   results[0]=0;
}

void loop()
{
   if(idx >= numberOfEntries)
   {
      Serial.println("Durations in Microseconds are:") ;
      for( byte i=0; i < numberOfEntries; i++)
      {
         Serial.print(i); Serial.print(": ");
         Serial.println(results[i]);
      }
      idx = 0; // Начинаем процесс анализирования снова
   }
   delay(1000);
}

void analyze()
{
```

```
    if(idx < numberOfEntries)
    {
       if(idx > 0)
       {
          results[idx] = micros() - microseconds;
       }
       idx = idx + 1;
    }
    microseconds = micros();
}
```

В качестве источника импульсов можно использовать модуль инфракрасного приемника, подключив его, как показано на рис. 10.1, и измеряя длительность импульсов, передаваемых инфракрасным пультом дистанционного управления. В качестве источника импульсов можно также использовать поворотный энкодер, подключив его, как показано на рис. 6.16 или даже простую кнопку, подключив ее, как показано на рис. 5.3.

Обсуждение работы решения и возможных проблем

В функции `setup()` оператор:

```
attachInterrupt(digitalPinToInterrupt(pin), analyze, CHANGE);
```

включает для скетча возможность обрабатывать прерывания. Первый аргумент, передаваемый функции, указывает прерывание, которое нужно инициализировать. Конкретное прерывание для определенного контакта зависит от используемой платы, поэтому при подключении прерывания к контакту вместо задания явного значения прерывания используется функция `digitalPinToInterrupt()`, которая определяет прерывание, подключаемое к этому контакту.

Следующий передаваемый функции аргумент указывает функцию, вызываемую для обслуживания прерывания, которая также называется *обработчиком прерывания*. В нашем случае это функция `analyze()`.

А в последнем аргументе указывается событие, активирующее прерывание. В нашем случае это ключевое слово `CHANGE`, которое задает как условие прерывания изменение уровня на контакте (т. е. переход от низкого состояния к высокому или наоборот). Возможны еще следующие три опции:

- `HIGH` — задает как условие прерывания высокий уровень на контакте (только для плат Due, Zero и MKR ARM);
- `LOW` — задает как условие прерывания низкий уровень на контакте;
- `RISING` — задает как условие прерывания переход от низкого уровня к высокому;
- `FALLING` — задает как условие прерывания переход от высокого уровня к низкому.

При чтении кода, который использует прерывания, имейте в виду, что изменение значений в скетче может быть неочевидным, поскольку обработчик прерывания не

вызывается напрямую в скетче, а вызывается тогда, когда возникает условие прерывания.

В рассматриваемом скетче в главном цикле `loop()` проверяется значение переменной `index`, чтобы определить, заполнил ли обработчик прерывания все элементы массива `results`. Но ничто в цикле `loop()` не меняет значение переменной `index`. Ее значение меняется в функции `analyze()`, когда возникает условие прерывания (меняется состояние контакта 2). Значение переменной `index` используется для указания следующего элемента массива `results`, в который нужно записать значение времени, прошедшее после последнего изменения состояния контакта. Время в микросекундах вычисляется вычитанием времени последнего изменения из текущего времени. Затем текущее время сохраняется как время последнего изменения состояния (в *главе 12* описывается этот метод для вычисления прошедшего времени с использованием функции `millis()`. Здесь же используется функция `micros()`, чтобы получить истекшее время в микросекундах вместо миллисекунд).

Переменные, которые изменяются в функции обработчика прерывания, объявляются как `volatile`. Если не использовать это ключевое слово `volatile`, то компилятор может сохранять эти значения в регистрах, которые могут быть случайно перезаписаны обработчиком прерывания. Чтобы не допустить этого, ключевое слово `volatile` дает указание компилятору сохранить соответствующие значения в памяти RAM, а не в регистрах.

При каждом активировании прерывания декрементируется значение переменной `index` и сохраняется текущее время. Также вычисляется разница между текущим временем и временем последнего предыдущего прерывания, и полученное значение сохраняется в соответствующем элементе массива `results` (за исключением первого активирования прерывания, когда значение `index` равно 0). Когда происходит достаточное количество прерываний, чтобы заполнить все элементы массива значениями, начинает исполняться вложенный цикл `for` главного цикла `loop()`, который выводит все эти значения в окно монитора порта.

Дополнительная информация

В *разд. 6.12* приводится пример использования внешних прерываний для определения вращения поворотного энкодера.

18.3. Выполнение периодических действий

ЗАДАЧА

Требуется периодически выполнять какое-либо действие, но при этом не тратить вычислительные ресурсы на постоянную проверку, истек ли заданный период времени. Кроме этого, для задания требуемого периода нужно использовать какой-либо простой интерфейс.

РЕШЕНИЕ

Самый простой способ решения этой задачи — использование таймера, реализованного в какой-либо библиотеке. Например, создавать регулярно повторяющиеся импульсы может и библиотека uTimerLib. Библиотека устанавливается стандартным способом с помощью Менеджера библиотек среды Arduino IDE (подробно об установке библиотек рассказано в *главе 16*). В листинге 18.3 приводится скетч решения с использованием этой библиотеки. Скетч мигает встроенным светодиодом платы с частотой, задаваемой с помощью монитора порта.

Листинг 18.3. Скетч для мигания светодиодом с использованием таймера

```
/*
* Скетч Timer pulse with uTimerLib
* Мигает встроенным светодиодом с частотой, задаваемой через монитор порта
*/

#include "uTimerLib.h"

const int pulsePin = LED_BUILTIN;

int period = 100; // Период мигания в миллисекундах

volatile bool output = HIGH; // Состояние контакта управления светодиодом

void setup()
{
   pinMode(pulsePin, OUTPUT);
   Serial.begin(9600);
   TimerLib.setInterval_us(flash, period/2 * 1000L);
}

void loop()
{
   if( Serial.available())
   {
      int period = Serial.parseInt();
      if (period)
      {
         Serial.print("Setting period to "); Serial.println(period);
         TimerLib.setInterval_us(flash, period/2 * 1000L);
      }
   }
}

void flash()
{
   digitalWrite(pulsePin, output);
   output = !output; // Инвертируем выходной сигнал
}
```

Обсуждение работы решения и возможных проблем

Запустите монитор порта, введите в строку передачи требуемое значение периода мигания в миллисекундах и нажмите клавишу <Enter>, чтобы отправить это значение скетчу. Скетч считывает значения из принимаемых по последовательному интерфейсу данных, используя функцию `parseInt()`, а затем делит полученное значение на 2, чтобы получить длительность включенного и выключенного состояний светодиода (*период мигания* — это общее время включенного и выключенного состояния, поэтому наименьшее значение, которое можно использовать, это 2). При задании периода мигания следует иметь в виду, что для человеческого глаза очень частое мигание может выглядеть как состояние постоянного включения. Поскольку период в функции:

```
TimerLib.setInterval_us()
```

задается в микросекундах, чтобы получить значение периода в миллисекундах, значение периода умножается на 1000.

На платах с микроконтроллером Atmega (например, Arduino Uno) библиотека uTimerLib использует таймер Timer2, поэтому она будет мешать работе функции `analogWrite()` на контактах 3 и 11, а на платах Arduino Mega — на контактах 9 и 10. На момент подготовки этой книги библиотека uTimerLib не поддерживает платы Arduino Uno WiFi Rev2 и Arduino Nano Every, в которых используется микроконтроллер Mega AVR, имеющий другую архитектуру таймеров. Но библиотека поддерживает несколько платформ на 32-разрядных микроконтроллерах, включая микроконтроллеры SAM (плата Arduino Due), SAMD21 (плата Arduino Zero и платы M0 компаний Adafruit и SparkFun), ESP8266 и др.

Библиотека uTimerLib позволяет использовать таймер, предоставляя временной интервал и название функции, подлежащей вызову по истечении этого интервала, как показано в следующей строке кода, осуществляющей установку таймера:

```
TimerLib.setInterval_us(flash, period/2 * 1000L);
```

В первом параметре функции `setInterval()` передается название функции, которую нужно исполнить по истечении заданного интервала. В нашем случае это функция `flash()`. А во втором параметре передается время таймера в микросекундах.

Подобно скетчу решения из *разд. 18.2*, скетч этого решения не вызывает функцию обработчика напрямую. Светодиод включается и выключается в функции `flash()`, которая вызывается таймером при каждом его достижении предельного значения счета. Код в главном цикле `loop()` только получает по последовательному интерфейсу сообщения со значением временного интервала и изменяет параметры таймера соответствующим образом.

Подход с использованием библиотеки позволяет реализовать управление таймером намного легче, чем посредством обращения к регистрам напрямую. Рассмотрим процесс внутренней работы этой библиотеки на ATmega. Таймеры постоянно ведут счет до заданного значения, подают сигнал по достижении этого значения, а затем повторяют процесс. Каждый таймер использует коэффициент предварительного деления частоты, который определяет частоту счета. Коэффициент предварительно-

ного деления частоты используется для деления временной базы (сигнала тактирования) системы на 1, 8, 64, 256 или 1024. Чем меньше коэффициент предварительного деления частоты, тем выше частота счета и тем быстрее достигается максимальное значение счета. Таким образом, временной интервал счета таймера определяется комбинацией частоты счета и максимального значения счета. Будучи 8-разрядным, таймер Timer2 может вести счет от 0 до 255, после чего снова начинает счет с нуля (таймеры Timer1, 3, 4 и 5 плат Mega являются 16-разрядными и могут считать до максимального значения 65535).

На платах с микроконтроллером AVR для временных интервалов свыше 4095 микросекунд библиотека uTimerLib использует коэффициент предварительного деления, равный 64. На платах Arduino с рабочей частотой 16 МГц каждый цикл ЦПУ занимает 62,5 наносекунды, а когда эту частоту разделить на 64, то каждый цикл таймера будет занимать 4000 наносекунд (62,5 × 64 = 4000).

Дополнительная информация

Библиотеку uTimerLib можно загрузить с ее репозитория GitHub (**https://oreil.ly/Gl-BL**).

18.4. Задание периода и длительности импульса

ЗАДАЧА

Требуется запрограммировать плату Arduino для создания импульсов с управляемым периодом и длительностью.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 18.4. Скетч генерирует импульсы ШИМ на контакте 9 частотой от 1 Гц до 1 МГц с помощью таймера Timer1. Используемая в скетче библиотека TimerOne поддерживает только платы Arduino Uno, Mega, Leonardo и несколько плат Teensy. Список поддерживаемых контактов для конкретной платы приводится на веб-сайте библиотеки (**https://oreil.ly/oQTjQ**).

Листинг 18.4. Скетч для установки периода и длительности импульсов

```
/*
* Скетч Pulse width duration
* Задает период и длительность импульсов
*/

#include <TimerOne.h>

const int outPin = 9; // Номер контакта.
                      // На платах Teensy 3.x - контакт 3 или 4,
                      // на платах Arduino Mega — контакты 11-13
```

```
long period = 40;      // Период 40 микросекунд (25 кГц)
long width = 20;       // Длительность 20 микросекунд (коэффициент заполнения 50%)

void setup()
{
   Serial.begin(9600);
   pinMode(outPin, OUTPUT);
   Timer1.initialize(period);      // Инициализируем таймер Timer1,
                                   // 10000 микросекунд
   int dutyCycle = map(width, 0,period, 0,1023);
   Timer1.pwm(outPin, dutyCycle); // Сигнал ШИМ на выходной контакт
}

void loop()
{
}
```

Обсуждение работы решения и возможных проблем

Значение периода импульса может находиться в диапазоне от одной до одного миллиона микросекунд и устанавливается в начале скетча заданием значения переменной `period`. Длительность импульса может быть любым значением в микросекундах, меньшим, чем значение периода. Задается длительность импульса присваиванием требуемого значения переменной `width`.

В скетче используется библиотека TimerOne (**https://oreil.ly/ULGGe**), управляющая 16-разрядным таймером Timer1 (который считает от 0 до 65535). На плате Arduino Uno этот же таймер используется функцией `analogWrite()` для подачи ШИМ-сигнала на контакты 9 и 10, поэтому библиотеку TimerOne и функцию `analogWrite()` нельзя использовать одновременно для работы с этими контактами. Скетч генерирует импульс на контакте 9, период и длительность которого задаются значениями переменных `period` и `width` соответственно.

Константы `OCR1A` и `OCR1B` определены в базовом коде языка Arduino (OCR означает Output Compare Register, регистр для сравнения выходного значения).

На микроконтроллерах AVR библиотека TimerOne использует несколько регистров. Обычно скетчам нет надобности обращаться непосредственно к аппаратным регистрам, поскольку операции с ними скрываются командами языка Arduino. Но в тех случаях, когда необходимо обращаться непосредственно к аппаратным возможностям микроконтроллера, чтобы использовать функциональность, не поддерживаемую стандартными командами языка Arduino, возникает надобность работать напрямую с этими регистрами. Подробная информация об организации этих регистров и работе с ними приводится в справочном листке (datasheet) на микроконтроллер. Для плат Teensy с микроконтроллерами семейства ARM применимы более или менее те же высокоуровневые понятия, но архитектура ARM использует совершенно другую схему для генерирования импульсов.

Период импульса определяется регистром ICR1 (Input Compare Register for Timer1). В этот регистр загружается 16-разрядное значение, которое используется в качестве максимальной величины счета таймера. Когда счет таймера доходит до этого значения, выполняется сброс таймера, и счет снова начинается вестись с нуля. В скетче этого решения каждый шаг счета длится 1 микросекунду, и поскольку регистру ICR1 задано значение 1000, то длительность каждого цикла счета составляет 1000 микросекунд.

Регистр OCR1A (или OCR1B — в зависимости от используемого контакта) содержит значение, определяющее длительность импульса. Когда счет таймера доходит до этого значения (и таймер работает в ШИМ-режиме, как это происходит в нашем примере), на контакте устанавливается низкий выходной уровень. Например, если каждый шаг счета длится одну микросекунду, регистру ICR1 присвоено значение 1000, а регистру OCR1A — значение 100, то в течение 100 микросекунд на контакте будет удерживаться высокий уровень, а в течение остальных 900 микросекунд периода — низкий.

Длительность каждого шага счета определяется рабочей частотой микроконтроллера Arduino (обычно 16 МГц) и значением коэффициента предварительного деления, на который делится рабочая частота микроконтроллера. Например, при рабочей частоте 16 МГц и коэффициенте предварительного деления, равном 64, длительность каждого шага счета таймера будет четыре микросекунды.

Период импульса задается инициализацией таймера Timer1 соответствующим значением:

```
Timer1.initialize(period);
```

Длительность импульса (т. е. длительность его активной части) будет задаваться относительно этого периода. Длительность импульса можно задать косвенно, установив с помощью функции Timer1.pwm() значение коэффициента заполнения в диапазоне от 0 до 1023, как мы это делали в случае с использованием функции analogWrite() в *разд. 7.2*. В нашем случае разница состоит в том, что мы можем использовать таймер Timer1 для управления периодом. Таким образом, чтобы получить равноотстоящие друг от друга импульсы длительностью 20 микросекунд, нужно задать период величиной 40 микросекунд и коэффициент заполнения величиной 50 процентов. Скетч этого решения позволяет пользователю задавать период и длительность импульсов, а также использует функцию map() для вычисления значения в диапазоне от 0 до 1023, которое нужно передать функции Timer1.pwm(). Частота импульсов с периодом 40 микросекунд составляет 25 КГц (1000000/40).

Дополнительная информация

В *разд. «Дополнительная информация» разд. 18.0* приведены ссылки на справочные листки и другие источники информации для таймеров.

18.5. Создание генератора импульсов

ЗАДАЧА

Требуется с помощью платы Arduino генерировать импульсы, управляя их характеристиками через монитор порта.

РЕШЕНИЕ

Эта задача решается скетчем из листинга 18.5. Он представляет собой улучшенную версию скетча решения из *разд. 18.4*, которая позволяет задавать частоту, период, длительность и коэффициент заполнения через монитор порта.

Листинг 18.5. Скетч генератора импульсов с расширенным управлением характеристиками сигнала

```
/*
* Скетч Configurable Pulse Generator
*/

#include <TimerOne.h>

const char SET_PERIOD_HEADER = 'p';
const char SET_FREQUENCY_HEADER = 'f';
const char SET_PULSE_WIDTH_HEADER = 'w';
const char SET_DUTY_CYCLE_HEADER = 'c';
const int outPin = 9; // Номер контакта.
                      // На платах Teensy 3.x - контакт 3 или 4,
                      // на платах Arduino Mega — контакты 11-13.

long period = 40; // Период 40 микросекунд (25 кГц)
int duty = 512;   // Коэффициент заполнения в диапазоне от 0 до 1023.
                  // 512 представляет коэффициент заполнения 50%

void setup()
{
   Serial.begin(9600);
   pinMode(outPin, OUTPUT);
   Timer1.initialize(period); // Инициализируем таймер timer1,
                              // 1000 микросекунд
   Timer1.pwm(outPin, duty);  // Сигнал ШИМ на выходной контакт
}

void loop()
{
   processSerial();
}
```

```
void processSerial()
{
   static long val = 0;
   if (Serial.available())
   {
      val = Serial.parseInt(); // Определяем первое число
      if (val)
      {
         char ch = Serial.read();
         switch(ch)
         {
            case SET_PERIOD_HEADER:
               period = val;
               Serial.print("Setting period to ");
                        // Устанавливаем период величиной:
               Serial.println(period);
               Timer1.setPeriod(period);
               Timer1.pwm(outPin, duty);
               show();
               break;
            case SET_FREQUENCY_HEADER:
               if(val > 0)
               {
                  Serial.print("Setting frequency to ");
                           // Устанавливаем частоту импульсов равной:
                  Serial.println(val);
                  period = 1000000 / val;
                  Timer1.setPeriod(period);
                  Timer1.pwm(outPin, duty);
               }
               show();
               break;
            case SET_PULSE_WIDTH_HEADER:
               if( val < period && val > 0)
               {
                  long width = val;
                  Serial.print("Setting Pulse width to ");
                           // Задаем длительность импульса, равную:
                  Serial.println(width);
                  duty = map(width, 0,period, 0,1023);
                  Timer1.pwm(outPin, duty);
               }
               else
                  Serial.println("Pulse width too long for current period");
                              // Длительность импульса слишком большая
                              // для заданного периода
               show();
               break;
```

```
            case SET_DUTY_CYCLE_HEADER:
               if( val >0 && val < 100)
               {
                  Serial.print("Setting Duty Cycle to ");
                            // Задаем коэффициент заполнения, равный:
                  Serial.println(val);
                  duty = map(val, 0,99, 0,1023);
                  Timer1.pwm(outPin, duty);
                  show();
               }
         }
      }
   }
}

void show()
{
   Serial.print("The period is "); // Период составляет
   Serial.println(period);
   Serial.print("Duty cycle is "); // Коэффициент заполнения составляет
   Serial.print(map(duty, 0,1023, 0,99));
   Serial.println("%");
   Serial.println();
}
```

Обсуждение работы решения и возможных проблем

Этот скетч основан на скетче решения из *разд. 18.4* с той лишь разницей, что в него добавлен код для получения через последовательный интерфейс и последующей установки значений частоты, периода, длительности и коэффициента заполнения генерируемых импульсов. В *главе 4* рассматривается метод накопления значения переменной `val`, которое затем используется для требуемого параметра, определяемого буквой команды.

В скетч можно добавить функцию `instructions()` для вывода в окно монитора порта инструкций по управлению генератором импульсов (листинг 18.6). Вызов этой функции можно осуществлять в функции `setup()` или `show()`.

Листинг 18.6. Функция `instructions()` для вывода в окно монитора порта инструкций по управлению генератором импульсов

```
void instructions()
{
   Serial.println("Send values followed by one of the following tags:");
               // Задавайте значение параметра, указывая тип параметра
               // в конце значения одной из следующих букв:
   Serial.println(" p - sets period in microseconds"); // период
                                                       // в микросекундах
```

```
    Serial.println(" f - sets frequency in Hz");  // частота в герцах
    Serial.println(" w - sets pulse width in microseconds");
                    // Длительность импульса в микросекундах
    Serial.println(" c - sets duty cycle in %");
                     // Коэффициент заполнения в процентах
    Serial.println("\n(duty cycle can have one decimal place)\n");
                   // Значение коэффициента заполнения может иметь
                   // один десятичный знак
}
```

Дополнительная информация

Дополнительная информация по работе с таймерами приводится также в *разд. 18.4*.

В *разд. «Дополнительная информация» разд. 18.0* приведены ссылки на справочные листки и другие источники информации для таймеров.

18.6. Изменение частоты ШИМ-сигнала таймера

ЗАДАЧА

Требуется повысить или понизить частоту сигнала ШИМ, используемого с функцией `analogWrite()` (см. *главу 7*). Например, управляемый с помощью функции `analogWrite()` электродвигатель издает гудение, свидетельствующее о слишком высокой частоте ШИМ-сигнала, и ее нужно понизить. Или мультиплексируемые светодиоды светятся неравномерно, что указывает на слишком низкую частоту ШИМ-сигнала, и ее нужно повысить.

РЕШЕНИЕ

Изменить частоту ШИМ-сигнала можно, меняя значение в регистре. Значения регистров и соответствующие им частоты приведены в табл. 18.2. Однако это решение применимо только к платам Arduino с микроконтроллером Atmega (например, Arduino Uno), но не к платам с микроконтроллером семейства ARM.

***Таблица 18.2.** Значения регистров для управления частотой ШИМ-сигнала*

Таймер Timer0 (для плат Uno — контакты 5 и 6, для плат Mega — контакты 4 и 3)		
Значение регистра TCCR0B	**Коэффициент предварительного делителя частоты (делитель)**	**Частота**
32 (1)	1	62500
33 (2)	8	7812,5
34	**64**	**976,5625**
35	256	244,140625
36	1024	61,03515625

Таблица 18.2 *(окончание)*

Таймер Timer1 (для плат Uno — контакты 9 и 10, для плат Mega — контакты 11 и 12)		
Значение регистра TCCR1B	**Коэффициент предварительного делителя частоты (делитель)**	**Частота**
1	1	312500
2	8	3906,25
3	**64**	**488,28125**
4	256	122,0703125
5	1024	30,517578125
Таймер Timer2 (для плат Uno — контакты 11 и 3, для плат Mega — контакты 9 и 10)		
Значение регистра TCCR12B	**Коэффициент предварительного делителя частоты (делитель)**	**Частота**
1	1	312500
2	8	3906,25
3	**64**	**488,28125**
4	256	122,0703125
5	1024	30,517578125

Все частоты указаны в герцах, предполагая рабочую частоту 16 МГц. Коэффициент предварительного деления частоты по умолчанию (64) выделен полужирным шрифтом.

Скетч решения приводится в листинге 18.7. Он позволяет устанавливать частоту таймера с помощью монитора порта. Частота задается вводом значения от 1 до 7, соответствующего требуемой частоте из правого столбца табл. 18.2, за которым следует буква (английская), указывающая требуемый таймер: *a* — для таймера Timer0, *b* — для таймера Timer1 и *c* — для таймера Timer2.

Листинг 18.7. Скетч, устанавливающий частоту ШИМ-сигнала для функции `analogWrite()`

```
/*
* Скетч Set PWM Frequency.
* Позволяет задавать частоту ШИМ-сигнала через монитор порта
* Частота задается цифрой от 1 до 7, за которой следует буква a, b или c,
  обозначающая таймер Timer0, Timer1, Timer2 соответственно.
*/

const byte mask = B11111000; // Маскируем биты, не входящие в значение
                             // коэффициента предварительного делителя
```

```
int prescale = 0;

void setup()
{
   Serial.begin(9600);
   analogWrite(3,128);
   analogWrite(5,128);
   analogWrite(6,128);
   analogWrite(9,128);
   analogWrite(10,128);
   analogWrite(11,128);
}

void loop()
{
   if (Serial.available())
   {
      char ch = Serial.read();
      if(ch >= '1' && ch <= '7') // Значение переменной ch действительная цифра?
      {
         prescale = ch - '0';
      }
      else if(ch == 'a' && prescale) // timer 0;
      {
         TCCR0B = (TCCR0B & mask) | prescale;
      }
      else if(ch == 'b' && prescale) // timer 1;
      {
         TCCR1B = (TCCR1B & mask) | prescale;
      }
      else if(ch == 'c' && prescale) // timer 2;
      {
         TCCR2B = (TCCR2B & mask) | prescale;
      }
   }
}
```

Следует избегать изменения частоты таймера `Timer0` (он используется функцией `analogWrite()` на контактах 5 и 7), поскольку это вызовет неправильную работу функций `delay()` и `millis()`.

Обсуждение работы решения и возможных проблем

Если при исполнении этого скетча к аналоговым контактам платы Arduino подключены только светодиоды, изменение частоты подаваемого на них ШИМ-сигнала не

вызовет заметного изменения их яркости. Это объясняется тем, что мы просто изменяем частоту их включения и выключения, а не отношение времени включенного состояния к выключенному. Этот и другие аспекты ШИМ-сигналов рассматриваются более подробно во *введении* в *главу 7* (*разд. 7.0*).

Частота ШИМ-сигнала таймера меняется при записи соответствующего значения в регистр `TCCRnB`, где буква *n* обозначает номер регистра, управляющий этим таймером. Платы Mega также имеют регистры `TCCR3B`, `TCCR4B` и `TCCR5B` для таймеров 3, 4 и 5.

Все выходные аналоговые (ШИМ) контакты таймера используют одну и ту же частоту, поэтому изменение частоты таймера отразится на всех контактах этого таймера.

Дополнительная информация

В *разд. «Дополнительная информация» разд. 18.0* приведены ссылки на справочные листки и другие источники информации для таймеров.

Платы Teensy 3.x поддерживают функцию `analogWriteFrequency()`, которая позволяет задавать частоту ШИМ в герцах. Дополнительная информация на этот счет приведена на веб-странице **https://oreil.ly/b_57C** сайта PJRC.

18.7. Подсчет импульсов

ЗАДАЧА

Требуется вести счет импульсов, поступающих на контакт, полностью аппаратными средствами, без расходования какого бы то ни было времени на программную обработку.

РЕШЕНИЕ

Эту задачу можно решить, используя счетчик импульсов, встроенный в схему таймера Timer1. Однако это решение применимо только к платам Arduino с микроконтроллером Atmega (например, Arduino Uno). Скетч решения приводится в листинге 18.8.

Листинг 18.8. Подсчет входящих импульсов аппаратными средствами

```
/*
* Скетч HardwareCounting
*
* Использует контакт 5 на платах с микроконтроллером ATmega168/328
*/

const int hardwareCounterPin = 5; // Входной контакт, подключенный
                                  // к внутреннему таймеру
```

```
const int ledPin = LED_BUILTIN;
const int samplePeriod = 1000;     // Период выборки в миллисекундах

unsigned int count;

void setup()
{
   Serial.begin(9600);
   pinMode(ledPin,OUTPUT);
   // Настройка аппаратного счетчика (подробная информация приводится
   // в справочном листке на микроконтроллеры ATmega)
   TCCR1A=0; // Сбрасываем регистр А управления таймером/счетчиком
}

void loop()
{
   digitalWrite(ledPin, LOW);
   delay(samplePeriod);
   digitalWrite(ledPin, HIGH);
   // Начинаем подсчет
   bitSet(TCCR1B, CS12); // Тактирующий сигнал для таймера берется
                         // с внешнего контакта
   bitSet(TCCR1B, CS11); // Тактирование по положительному перепаду
   delay(samplePeriod);
   // Прекращаем подсчет
   TCCR1B = 0;
   count = TCNT1;
   TCNT1 = 0; // Обнуляем аппаратный счетчик
   if(count > 0)
      Serial.println(count);
}
```

Обсуждение работы решения и возможных проблем

Чтобы протестировать работу скетча, подключите контакт приема последовательного интерфейса (контакт 0) к контакту ввода (контакт 5 на плате Arduino Uno). Каждый передаваемый символ должен вызывать увеличение счета, при этом конкретная величина этого увеличения зависит от количества импульсов, необходимых для представления значения ASCII-символа (имейте в виду, что импульсы собственно символов, передаваемых по последовательному интерфейсу, заключены между стартовым и стоповым импульсами). Вот несколько примеров интересных последовательностей импульсов ряда символов:

```
'u' = 01010101
'3' = 00110011
'~' = 01111110
'@' = 01000000
```

При наличии двух плат Arduino, одну из них можно использовать в качестве генератора импульсов, исполняя на ней скетч генератора импульсов, рассмотренный в одном из предыдущих решений этой главы, и подключить контакт вывода импульсов (контакт 9) этой платы к контакту ввода (контакт 5) платы, на которой исполняется скетч подсчета количества импульсов. Генератор импульсов также использует таймер Timer1 (это единственный 16-разрядный таймер на плате Uno и других платах с микроконтроллером ATmega328), что позволяет совместить эту функциональность, используя одну плату.

При аппаратном подсчете импульсов используется контакт, который внутри платы подключен к счетчику аппаратным способом, и его нельзя заменить каким-либо другим контактом. Для платы Arduino Uno это контакт 5, а в платах Mega используется таймер Timer5, подключенный к контакту 47. Для этих плат названия регистров `TCCR1A` и `TCCR1B` нужно заменить названиями `TCCR5A` и `TCCR5B` соответственно.

Регистр `TCCR1B` таймера управляет режимом счета — запись в этот регистр значения 0 останавливает счет. Значения, используемые в коде в главном цикле `loop()`, разрешают счет по положительному перепаду импульсов на входном контакте. Регистр `TCNT1` таймера Timer1 объявлен в базовом коде языка Arduino и служит для аккумулирования значения счета.

Текущее значение подсчета отображается каждую секунду кодом в цикле `loop()`. Если на контакте 5 не обнаруживается никаких импульсов, то будет выводиться значение 0.

Дополнительная информация

Рассмотренный в этом решении метод подсчета импульсов используется в библиотеке FreqCount (**https://oreil.ly/wBTES**).

Для подсчета количества изменений состояния контакта (т. е. для подсчета импульсов) также можно использовать прерывания. См. подробности в *разд. 18.2*.

В *разд. «Дополнительная информация» разд. 18.0* приведены ссылки на справочные листки и другие источники информации для таймеров.

18.8. Измерение характеристик импульсов с более высокой точностью

ЗАДАЧА

Требуется измерять период или длительность включенного или выключенного состояния импульсов. Измерения должны быть максимально точными, поэтому нельзя допускать никаких задержек с вызовом обработчика прерывания (как это было в решении из *разд. 18.2*), поскольку это отрицательно скажется на точности измерений.

РЕШЕНИЕ

Эту задачу можно решить, используя аппаратную возможность измерения импульсов, встроенную в схему таймера Timer1. В решении задействуются средства, специфичные для микроконтроллеров AVR, поэтому оно применимо только к платам с микроконтроллером Atmega (например, Arduino Uno). Скетч решения приводится в листинге 18.9.

Листинг 18.9. Скетч для измерения характеристик импульсов с высокой точностью

```
/*
* Скетч InputCapture
* Использует аппаратный таймер для точного измерения характеристик
  импульсов, поступающих на контакт 8 платы Arduino
  с микроконтроллером ATmega168/328
*/

/* Несколько примеров интересных последовательностей импульсов
   ряда символов:
u 01010101
3 00110011
~ 01111110
@ 01000000
*/


const int inputCapturePin = 8;  // Входной контакт, подключенный
                                // к внутреннему таймеру
const int ledPin = LED_BUILTIN;
const int prescale = 8;         // Коэффициент предварительного делителя
                // частоты (при рабочей частоте 16 МГц каждый такт длится 0,5 мкс)
const byte prescaleBits = B010; // См. табл. 18.1 или справочный листок

// Вычисляем длительность такта в наносекундах
const long precision = (1000000/(F_CPU/1000.0)) * prescale;
const int numberOfEntries = 64;    // Максимальное количество импульсов для измерения
const int gateSamplePeriod = 1000; // Период выборки в миллисекундах

volatile byte index = 0; // Индекс к сохраненным данным
volatile byte gate = 0;  // 0 отключает захват, 1 – включает
volatile unsigned int results[numberOfEntries]; // Обратите внимание на то,
                                                // что это 16-разрядное значение

/* Вектор прерывания ICR */
ISR(TIMER1_CAPT_vect)
{
   TCNT1 = 0; // Обнуляем аппаратный счетчик
```

```
   if(gate)
   {
      if( index != 0 || bitRead(TCCR1B ,ICES1) == true) // Ожидаем
                                                        // положительный перепад
      {
         // Обнаружен отрицательный перепад
         if(index < numberOfEntries)
         {
            results[index] = ICR1; // Сохраняем захваченное входное значение
            index++;
         }
      }
   }
   TCCR1B ^= _BV(ICES1); // Переключаем бит для активирования захвата
                         // по противоположному перепаду
}

void setup()
{
   Serial.begin(9600);
   pinMode(ledPin, OUTPUT);
   pinMode(inputCapturePin, INPUT); // Контакт захвата ввода ICP (цифровой
                          // контакт 8 на платах Arduino) в качестве ввода
   TCCR1A = 0 ;              // Обычный режим счета
   TCCR1B = prescaleBits ;   // Устанавливаем биты предварительного делителя
   TCCR1C = 0;
   bitSet(TCCR1B,ICES1);     // Инициализируем захват ввода
   bitSet(TIFR1,ICF1);       // Очищаем
   bitSet(TIMSK1,ICIE1);     // Включаем
   Serial.println("pulses are sampled while LED is lit");
               // Импульсы считываются, пока горит светодиод
   Serial.print( precision); // Выводим на экран длительность каждого
                             // такта в микросекундах
   Serial.println(" microseconds per tick"); // микросекунд на такт
}

// Код цикла loop() выводит в окно монитора порта длительность
// импульсов, считанных в последнюю секунду
void loop()
{
   digitalWrite(ledPin, LOW);
   delay(gateSamplePeriod);
   digitalWrite(ledPin, HIGH);
   index = 0;
   gate = 1; // Включаем выборку
   delay(gateSamplePeriod);
   gate = 0; // Выключаем выборку
```

```
    if(index > 0)
    {
      Serial.println("Durations in Microseconds are:") ;
                  // Длительность импульсов в микросекундах:
      for( byte i=0; i < index; i++)
      {
        long duration;
        duration = results[i] * precision; // Длительность импульса
                                           // в наносекундах
        if(duration > 0)
        {
          Serial.println(duration / 1000); // Длительность импульсов
                                           // в микросекундах
          results[i] = 0; // Очищаем значения для следующей выборки
        }
      }
      index = 0;
    }
}
```

Обсуждение работы решения и возможных проблем

Для измерения длительности импульсов в скетче используется функциональность захвата ввода `Input Capture`. Эта возможность поддерживается лишь 16-разрядными таймерами и, кроме того, работает только на контакте 8 платы Arduino Uno или совместимых плат.

Функциональность `Input Capture` использует контакт, который внутри платы подключен к счетчику аппаратным способом, и его нельзя заменить каким-либо другим контактом. На плате Uno это контакт 8, а на плате Mega — контакт 48 (и таймер Timer5 вместо таймера Timer1).

Поскольку функциональность `Input Capture` реализована полностью аппаратными средствами микроконтроллера платы, на обработку прерываний не тратится никакого времени, вследствие чего этот подход дает более точные результаты измерений характеристик очень коротких импульсов (длящихся меньше десятков микросекунд).

В скетче используется переменная `gate`, которая разрешает измерения (ненулевых значений) через секунду. Активность процесса измерения обозначается включенным светодиодом. Обработчик прерывания захвата ввода сохраняет длительность импульса вплоть до 64 перепадов импульса.

Перепад, активирующий измерение, определяется битом `ICES1` регистра `TCCR1B` таймера. Перепад, активирующий обработчик, меняется на обратный оператором:

```
TCCR1B ^= _BV(ICES1);
```

что позволяет измерять длительность импульсов как высокого, так и низкого уровня.

Если счет превышает максимальное значение счетчика, такое переполнение можно отслеживать, чтобы инкрементировать переменную для расширения диапазона счета. Это реализуется в следующем фрагменте кода, который инкрементирует переменную `overflow` при каждом переполнении счетчика:

```
volatile int overflows = 0;
/* Вектор прерывания переполнения */
ISR(TIMER1_OVF_vect) // Переходим сюда, если не обнаружено импульсов
{
  overflows++; // Инкрементируем счетчик переполнений
}
```

Для использования этого кода нужно модифицировать код в функции `setup()` следующим образом:

```
TIMSK1 = _BV(ICIE1);    // Разрешаем прерывание захвата ввода
                        // для таймера Timer1
TIMSK1 |= _BV(TOIE1);   // Добавьте эту строку кода, чтобы разрешить
                        // прерывание по переполнению
```

Дополнительная информация

В *разд. «Дополнительная информация» разд. 18.0* приведены ссылки на справочные листки и другие источники информации для таймеров.

18.9. Оперативное измерение аналоговых значений

ЗАДАЧА

Требуется максимально быстро считывать значение входящего аналогового сигнала, не теряя при этом точности измерения.

РЕШЕНИЕ

Частоту выборки функции `analogRead()` можно повысить, записав соответствующее значение в регистр, управляющий частотой выборки. Соответствующий скетч приводится в листинге 18.10. Скетч будет работать только на платах с микроконтроллером Atmega (например, Arduino Uno).

Листинг 18.10. Скетч для повышения частоты выборки функции `analogRead()`

```
/*
* Скетч Sampling rate
* Повышает частоту выборки функции analogRead()
*/

const int sensorPin = 0;          // Контакт, к которому подключен приемник
const int numberOfEntries = 100;
```

```
unsigned long microseconds;
unsigned long duration;

int results[numberOfEntries];

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Требуется для плат Leonardo

   // Стандартные характеристики функции analogRead() (предварительный делитель = 128)
   microseconds = micros();
   for(int i = 0; i < numberOfEntries; i++)
   {
      results[i] = analogRead(sensorPin);
   }
   duration = micros() - microseconds;
   Serial.print(numberOfEntries);
   Serial.print(" readings took ");
   Serial.println(duration);
   // работа с высокой частотой сигнала тактирования (устанавливаем
   // предварительный делитель, равный 16)
   bitClear(ADCSRA,ADPS0);
   bitClear(ADCSRA,ADPS1);
   bitSet(ADCSRA,ADPS2);
   microseconds = micros();
   for(int i = 0; i < numberOfEntries; i++)
   {
      results[i] = analogRead(sensorPin);
   }
   duration = micros() - microseconds;
   Serial.print(numberOfEntries);
   Serial.print(" readings took ");
   Serial.println(duration);
}

void loop()
{
}
```

В результате исполнения этого скетча на плате Arduino с рабочей частотой 16 МГц в окне монитора порта отобразится следующий вывод:

```
100 readings took 11308 (100 считываний заняло 11308)
100 readings took 1704
```

Обсуждение работы решения и возможных проблем

Для считывания значения входного аналогового сигнала функции `analogRead()` требуется около 110 микросекунд. Этого может быть недостаточно для считывания быстро меняющихся значений — например, высшего диапазона звуковых частот. Скетч решения измеряет время в микросекундах для стандартной операции чтения функции `analogRead()`, а затем меняет частоту временной базы, используемую аналого-цифровым преобразователем (АЦП), чтобы выполнить преобразование быстрее. Для платы с рабочей частотой 16 МГц частота временной базы повышается с 125 кГц до 1 МГц. Но производительность фактически повышается чуть меньше, чем в восемь раз, поскольку повышение частоты временной базы не влияет на некоторые накладные издержки функции `analogRead()`. Тем не менее уменьшение времени операции со 113 микросекунд до 17 микросекунд является значительным улучшением производительности.

Для настройки АЦП используется регистр `ADCSRA`, а установкой в скетче битов `ADPS0`, `ADPS1` и `ADPS2` для делителя сигнала тактирования задается значение 16.

Дополнительная информация

В документе по применению компании Microchip (**https://oreil.ly/lrr3s**) приведено подробное объяснение аспектов производительности аналого-цифрового преобразователя.

На платах Arduino с микроконтроллером архитектуры ARM (например, Zero и совместимых с ней плат) функция `analogRead()` работает значительно медленнее, чем на платах с микроконтроллером AVR. Библиотека AnalogReadFast позволяет повысить скорость считывания аналогового сигнала приблизительно в пять раз по сравнению со стандартной функцией `analogRead()` на платах с микроконтроллером AVR и приблизительно в 20 раз быстрее, чем на платах с микроконтроллером SAMD21 (например, Arduino Zero). Библиотеку можно загрузить на ее веб-странице (**https://oreil.ly/sz7C9**) сайта GitHub. Устанавливается она стандартным образом — с помощью Менеджера библиотек.

18.10. Понижение энергопотребления приложения

ЗАДАЧА

Требуется понизить энергопотребление приложения, отключив плату Arduino на некоторый период времени, или пока не произойдет определенное внешнее событие.

РЕШЕНИЕ

Эта задача решается скетчем, код которого приводится в листинге 18.11. В скетче решения используется библиотека SleepyDog компании Adafruit, которая поддерживает платы Arduino Uno, Mega, Zero, Leonardo, платы M0 компании Adafruit и частично платы Teensy 3.x.

Листинг 18.11. Скетч для уменьшения энергопотребления приложения

```
/*
* Скетч Low power
* Понижает энергопотребление приложения, используя библиотеку SleepyDog
  компании Adafruit
*/

#include <Adafruit_SleepyDog.h>

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

void loop()
{
   digitalWrite(LED_BUILTIN, HIGH);
   int sleepTimeMillis1 = Watchdog.sleep(500);
   digitalWrite(LED_BUILTIN, LOW);
   int sleepTimeMillis2 = Watchdog.sleep(500);
   // Пытаемся восстановить USB-подключение на плате Leonardo и других
   // платах со встроенной поддержкой USB
   #if defined(USBCON) && !defined(USE_TINYUSB)
   USBDevice.attach();
   #endif
   Serial.print("Slept for "); Serial.print(sleepTimeMillis1);
   Serial.print("ms and "); Serial.print(sleepTimeMillis2);
   Serial.println("ms");
   delay(100); // Даем буферу последовательного интерфейса достаточное
               // время для выполнения передачи
}
```

При исполнении этого скетча на платах с микроконтроллером семейства SAMD21 (например, плате Arduino Zero или платах серии MKR) после первого отключения питания могут возникнуть проблемы с восстановлением подключения USB. Если потребуется перезагрузить скетч в плату, то нужно выполнить быстрое двойное нажатие кнопки сброса, чтобы перевести плату в режим загрузчика (в этом режиме скетч не будет исполняться).

Обсуждение работы решения и возможных проблем

При постоянной работе платы Arduino Uno 9-вольтовая щелочная батарейка разрядится через несколько недель. Энергопотребление приложения можно значительно понизить, если приостановить его работу на некоторый период времени, что можно

сделать благодаря возможностям платы Arduino переходить в режим сна на заданный период времени. В этом режиме энергопотребление микроконтроллера понижается до менее чем одной сотой процента обычного уровня энергопотребления (с приблизительно 15 мА до приблизительно 0,001 мА).

Доступ к функции перехода в режим сна предоставляется используемой в скетче решения библиотекой SleepyDog. Время сна можно задать в диапазоне от 15 до 8000 миллисекунд (8 секунд). Для более длительных периодов сна можно повторять стандартный максимальный период, пока не получиться период требуемой длительности. В листинге 18.12 приводится код функции, реализующей этот подход.

Листинг 18.12. Функция для получения более длительных периодов сна, чем максимальный стандартный

```
unsigned long longDelay(long milliseconds)
{
   unsigned long sleptFor = 0;
   while(milliseconds > 0)
   {
      if(milliseconds > 8000)
      {
         milliseconds -= 8000;
         sleptFor += Watchdog.sleep(8000);
      }
      else
      {
         sleptFor += Watchdog.sleep(milliseconds);
         break;
      }
   }
   return sleptFor;
}
```

В каждом цикле указанный период сна сводится к ближайшему периоду, поддерживаемому той или иной аппаратной платформой. Например, если для платы с микроконтроллером AVR указать период сна длительностью 8600 миллисекунд, то в первой итерации цикла будет реализован период сна длительностью 8000 миллисекунд. Но в следующей итерации будет реализован период сна длительностью 500 миллисекунд, поскольку для микроконтроллеров AVR это первый действительный период сна перед следующим более длительным (1000 миллисекунд). Конкретные примеры реализации более длительных периодов сна приводятся в файлах *.cpp, расположенных в папке utility библиотеки SleepyDog (**https://oreil.ly/taz_*O***).

Хотя использование режима сна может способствовать понижению энергопотребления микроконтроллером платы Arduino, для максимального продления времени

работы на батарейке следует минимизировать потребление тока внешними компонентами платы — например, некачественными стабилизаторами питания, понижающими или повышающими резисторами, светодиодами и другими компонентами, потребляющими ток даже тогда, когда микроконтроллер находится в режиме сна.

Дополнительная информация

Библиотека narcoleptic (**https://oreil.ly/Rrkp1**) позволяет реализовать переход микроконтроллера в режим сна на заданный период времени, а также до поступления сигнала на определенный контакт платы. Но эта библиотека поддерживает только платы с микроконтроллерами архитектуры AVR.

Библиотека ArduinoLowPower поддерживает режим сна на платах с микроконтроллерами семейства SAMD, а также на платах с микроконтроллером NRF52 — таких как платы Nano 33 BLE и Nano 33 BLE Sense. Библиотека устанавливается стандартным образом — с помощью Менеджера библиотек. Дополнительная информация об этой библиотеке приводится на ее веб-странице (**https://oreil.ly/e-nil**) сайта Arduino.

На веб-странице Lab3 (**https://oreil.ly/agdmy**) приведен пример работы с очень малым энергопотреблением.

18.11. Быстрая установка уровней на цифровых контактах

ЗАДАЧА

Требуется устанавливать высокие и низкие уровни на цифровых контактах намного быстрее, чем это позволяет делать стандартная функция языка Arduino `digitalWrite()`.

РЕШЕНИЕ

Стандартная функция `digitalWrite()` языка Arduino предоставляет легкий и безопасный способ установки высокого или низкого уровня на цифровых контактах, но делает она это более чем в 30 раз медленнее, чем подход с прямым доступом к соответствующему аппаратному обеспечению микроконтроллера. В частности, устанавливать требуемые уровни на цифровых контактах можно, устанавливая или очищая биты аппаратных регистров, которые управляют соответствующими цифровыми контактами.

Скетч такого решения приводится в листинге 18.13. Скетч использует прямой доступ к аппаратному обеспечению ввода/вывода для передачи радиосигнала со словом «arduino» в коде Морзе, который можно прослушивать по радио, настроенному на частоту приблизительно 1 МГц. Этот подход в 30 раз быстрее, чем использование стандартной функции `digitalWrite()` языка Arduino.

Листинг 18.13. Скетч для задания уровней на цифровых контактах с помощью установки битов регистров управления контактами

```
/*
* Скетч Morse
*
* Использование прямого управления цифровым контактом ввода/вывода
  для передачи сигнала AM на частоте 1 МГц
*/

const int sendPin = 2;
const byte WPM = 12;              // Скорость передачи в словах в минуту
const long repeatCount = 1200000 / WPM; // Счет определяет длительность точки/тире
const byte dot = 1;
const byte dash = 3;
const byte gap = 3;
const byte wordGap = 7;byte letter = 0; // Передаваемая буква

char *arduino = ".- .-. -.. ..- .. -. ---";

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

void loop()
{
   sendMorse(arduino);
   delay(2000);
}

void sendMorse(char * string)
{
   letter = 0 ;
   while(string[letter]!= 0)
   {
      if(string[letter] == '.')
      {
         sendDot();
      }
      else if(string[letter] == '-')
      {
         sendDash();
      }
      else if(string[letter] == ' ')
      {
         sendGap();
      }
```

```
      else if(string[letter] == 0)
      {
         sendWordGap();
      }
      letter = letter+1;
   }
}

void sendDot()
{
   transmitCarrier(dot * repeatCount);
   sendGap();
}

void sendDash()
{
   transmitCarrier(dash * repeatCount);
   sendGap();
}

void sendGap()
{
   transmitNoCarrier(gap * repeatCount);
}

void sendWordGap()
{
   transmitNoCarrier(wordGap * repeatCount);
}

void transmitCarrier(long count)
{
   while(count--)
   {
      bitSet(PORTD, sendPin);
      bitSet(PORTD, sendPin);
      bitSet(PORTD, sendPin);
      bitSet(PORTD, sendPin);
      bitClear(PORTD, sendPin);
   }
}

void transmitNoCarrier(long count)
{
   while(count--)
   {
      bitClear(PORTD, sendPin);
```

```
        bitClear(PORTD, sendPin);
        bitClear(PORTD, sendPin);
        bitClear(PORTD, sendPin);
        bitClear(PORTD, sendPin);
    }
}
```

Подключите к контакту 2 платы Arduino, на которой исполняется этот скетч, один конец провода, а другой его конец разместите вблизи антенны радиоприемника среднечастотного диапазона, настроенного на частоту 1 МГц (1000 кГц).

Обсуждение работы решения и возможных проблем

Скетч генерирует радиосигнал частотой 1МГц, модулированный точками и тире, который можно принимать на любом радиоприемнике со среднечастотным диапазоном, настроенным на эту частоту. Частота радиосигнала определяется длительностью высоких и низких уровней, устанавливаемых на контакте операторами `bitSet()` и `bitClear()` соответственно, генерируя таким образом радиосигнал. Операторы `bitSet()` и `bitClear()` представляют собой не функции, а макросы. Эти макросы заменяют выражение исполняемым кодом, которым в нашем случае является код, изменяющий один бит в регистре `PORTD`, задаваемый значением переменной `sendPin`.

Цифровые контакты ввода/вывода с 0 по 7 управляются регистром `PORTDD`. Каждый бит этого регистра соответствует одному цифровому контакту. Контакты с 8-го по 13-й управляются регистром `PORTB`, а контакты с 14-го по 19-й — регистром `PORTA`. В скетче используются операторы `bitSet()` и `bitClear()`, чтобы устанавливать (задавать значение 1) и сбрасывать (устанавливать значение 0) битов регистра. Каждый регистр содержит до восьми битов (хотя не все из этих битов соответствуют контактам платы Arduino). Чтобы, например, вместо контакта 2 платы Arduino использовать контакт 3, нужно работать с соответствующим битом регистра `PORTB`, как показано в следующем фрагменте кода:

```
const int sendPin = 13;

bitSet(PORTB, sendPin - 8);
bitClear(PORTB, sendPin - 8);
```

Значение 8 отнимается от номера контакта по той причине, что бит 0 регистра `PORTB` управляет контактом 8, бит 1 — контактом 9 и т. д., т. е. управляющий бит на 8 меньше номера управляемого контакта. Поэтому контактом 13 управляет бит 5.

Установка и сброс битов посредством операторов `bitSet()` и `bitClear()` выполняются за одну инструкцию микроконтроллера платы Arduino. Для плат Arduino с рабочей частотой 16 МГц одна инструкция исполняется за 62,5 наносекунды, что приблизительно в 30 раз быстрее, чем исполнение функции `digitalWrite()`.

Функциям `transmit` скетча, по сути, требуется больше времени для обновления и проверки переменной `count`, чем занимает установка и сброс битов регистра. Этим

объясняется наличие в функции `transmitCarrier()` четырех операторов `bitSet()` и только одного оператора `bitClear()` — дополнительные операторы `bitClear()` не нужны по той причине, что обновление и проверка переменной `count` занимают дополнительное время.

18.12. Загрузка скетчей с помощью программатора

ЗАДАЧА

Требуется загрузить скетч в плату Arduino с микроконтроллером семейства AVR (например, в плату Arduino Uno), используя аппаратный программатор, а не программу загрузчика микроконтроллера. Допустим, что нужно свести к минимуму время загрузки, или отсутствует последовательное подключение к компьютеру, подходящее для использования загрузчика, или надо увеличить объем программной памяти для загрузки скетча за счет памяти, используемой для загрузчика.

РЕШЕНИЕ

Подключите внешний ISP-программатор к разъему ICSP (In-Circuit Serial Programming, внутрисхемное последовательное программирование) платы Arduino. Программаторы, предназначенные для использования с платой Arduino, оснащены кабелем с 6-гнездовым разъемом, который подключается к 6-штыревому ICSP-разъему платы Arduino, как показано на рис. 18.1.

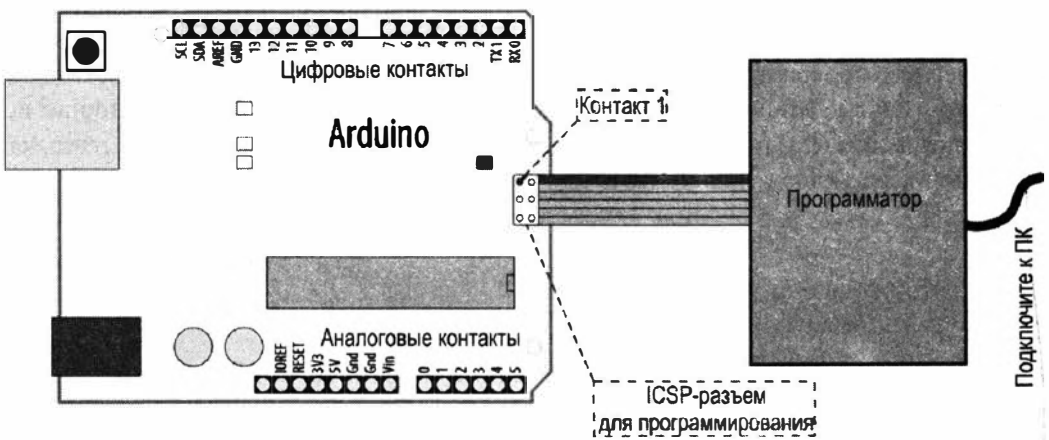


Рис. 18.1. Подключение внешнего программатора к плате Arduino

При подключении программатора обязательно убедитесь в том, что контакт 1 разъема программатора (который обычно обозначается подключенной к нему жилой шлейфа иного цвета, чем остальные жилы) подключен к контакту 1 ICSP-разъема платы Arduino. Программатор может быть оснащен выключателем или перемычкой, позволяющей использовать его в качестве источника питания для платы

Arduino. Внимательно изучите инструкции на используемый вами программатор, чтобы обеспечить правильную запитку платы Arduino.

В меню **Инструменты** среды Arduino IDE выберите тип используемого вами программатора (AVRISP, AVRISPII, USBtinyISP, Parallel programmer или Arduino as ISP) и заодно проверьте, что выбрана правильная плата. Затем в меню **Скетч** выберите опцию **Загрузить через программатор**, чтобы загрузить текущий скетч в плату Arduino.

Обсуждение работы решения и возможных проблем

На рынке предлагается несколько разных программаторов — в диапазоне от дорогостоящих устройств, оснащеных различными возможностями отладки и предназначенных для профессиональных разработчиков, до недорогих наборов для самостоятельной сборки. Кроме этого, в качестве программатора можно использовать одну плату Arduino для программирования другой. Программатор может быть натуральным USB-устройством или же отображаться в Менеджере устройств как последовательный порт. Изучите справочный листок на используемое вами устройство, чтобы разобраться, что оно собой представляет и нужно ли для него устанавливать драйверы.

При загрузке скетча в плату Arduino с помощью внешнего аппаратного программатора светодиоды Rx и Tx последовательного интерфейса не будут мигать, поскольку в этом случае последовательный порт не используется.

Загрузка скетча с помощью внешнего аппаратного программатора удаляет из программной памяти микроконтроллера программный загрузчик. Это освобождает программную память, занимаемую программным загрузчиком, которую можно использовать для загрузки скетчей большего объема. Чтобы восстановить программный загрузчик, выполните команду меню **Инструменты | Записать загрузчик** при подключенном программаторе ISP.

Дополнительная информация

В скетче примера ArduinoISP содержится код для использования платы Arduino в качестве ISP-программатора. Подробные инструкции по использованию ее в этом качестве приводятся на веб-странице **Arduino as ISP and Arduino Bootloaders** (Arduino в качестве ISP и загрузчики Arduino) сайта Arduino (**https://oreil.ly/f-C0p**). Кроме этого, подключение платы поясняется в комментариях в примере.

В *разд. 18.13* приводится более подробная информация по программному загрузчику Arduino.

В число подходящих аппаратных программаторов входят следующие:

- USBtinyISP (**https://oreil.ly/TsYk9**);
- AVRISP mkII компании Microchip (**https://oreil.ly/8J8Ym**).

18.13. Обновление или замена загрузчика Arduino

ЗАДАЧА

Требуется заменить загрузчик на плате с микроконтроллером семейства AVR (например, на плате Arduino Uno). Возможно, плата не загружает программы, и вы подозреваете, что что-то не в порядке с загрузчиком. Или вы хотите заменить старый загрузчик новым, более эффективным или обладающим дополнительными возможностями.

РЕШЕНИЕ

Подключите аппаратный программатор к плате Arduino, как описывается в *разд. 18.12*, и выберите его в списке программаторов. Удостоверьтесь в том, что выбрана правильная плата и из меню **Инструменты** выполните команду **Записать загрузчик**.

В консоли сообщений среды Arduino IDE отобразится сообщение **Burning bootloader to I/O board (this may take a minute)...** (Записывается загрузчик в плату ввода/вывода (это может занять некоторое время...)). На программаторах, оснащенных индикаторными светодиодами, должен гореть индикатор записи загрузчика в плату. При записи загрузчика в микроконтроллер платы Arduino Uno должен мигать ее встроенный светодиод (контакт 13, к которому подключен встроенный светодиод платы, также подключен к одному из сигнальных контактов разъема ICSP). В случае успешной записи в консоли сообщений среды Arduino IDE должно отобразиться сообщение **Done Loading Bootloader** (Загрузчик записан).

Отключите плату от программатора, подключите ее к компьютеру через USB-разъем и попытайтесь записать в нее скетч, используя среду Arduino IDE, чтобы проверить успешность записи загрузчика.

Обсуждение работы решения и возможных проблем

Загрузчик представляет собой небольшую программу, исполняющуюся на микроконтроллере платы при каждом включении питания или сброса микроконтроллера и проверяющую, не пытается ли среда Arduino IDE загрузить код в плату. Если да, то загрузчик продолжает исполняться и записывает в программную память код нового скетча, передаваемый через последовательный интерфейс, и после успешного завершения загрузки передает управление этому коду. Если же загрузчик не обнаруживает запроса на запись скетча, то он передает управление коду скетча, который уже загружен в плату.

В случае использования последовательного программатора после завершения записи загрузчика нужно переключить последовательный порт обратно на тот, который присвоен подключенной плате Arduino, как описывается в *разд. 1.4*.

Дополнительная информация

Обновить или заменить загрузчик можно также и с помощью скетча Optiloader (**https://oreil.ly/47ZV5**), разработанного Биллом Вестфилдом (Bill Westfield). Скетч содержит загрузчики для всех микроконтроллеров семейства AVR и использует плату Arduino в качестве ISP-программатора. Это дает возможность автоматически записывать загрузчик в микроконтроллер другой платы Arduino, не требуя подключения к компьютеру. Код автоматически определяет микроконтроллер и записывает в него соответствующий загрузчик.

Обновление Optiboot (**https://oreil.ly/RMXZt**) стандартного загрузчика Arduino увеличивает объем памяти для загрузки скетчей приложений, повышает скорость загрузки скетчей, а также поддерживает несколько других возможностей. Но этот загрузчик не записывается напрямую в память микроконтроллера. Чтобы это сделать, нужно выбрать требуемый пакет основных файлов Arduino на основе Optiboot (из списка пакетов, который приводится в файле Readme), установить его с помощью Менеджера плат, а затем записать загрузчик, выполнив команду **Записать загрузчик** меню **Инструменты** среды Arduino IDE.

18.14. Перемещение курсора мыши компьютера

ЗАДАЧА

Требуется взаимодействовать с приложением на компьютере, перемещая курсор мыши компьютера с помощью скетча Arduino. Возможно, вы хотите переместить курсор мыши в определенное место на экране в ответ на какие-либо данные. Например, к плате Arduino подключено устройство ввода — потенциометр или игровой контроллер Wii Nunchuk (см. *разд. 13.6*) — и с его помощью нужно управлять позицией курсора мыши в исполняющейся на компьютере программе.

РЕШЕНИЕ

Пример скетча решения этой задачи приводится в листинге 18.14. Скетч перемещает курсор мыши по экрану компьютера на основании данных от двух потенциометров. Скетч предназначен для работы на платах с микроконтроллерами архитектуры ARM (например, Arduino Zero, Metro M0 Express компании Adafruit и RedBoard Turbo компании SparkFun), а также на платах с микроконтроллерами ATmega32u4 (например, Arduino Leonardo). Благодаря встроенной библиотеке Arduino Mouse все эти платы могут определяться компьютером как мышь с USB-интерфейсом. На платах Arduino Uno и прямо совместимых с ними платах скетч работать не будет.

Для работы скетча к плате нужно подключить два потенциометра — к аналоговым контактам A4 и A5 (подробно о подключении потенциометра к плате Arduino рассказано в *разд. 5.6*). Также нужно подключить кнопку к цифровому контакту 2 (как показано в *разд. 5.2*). Эта кнопка будет играть роль левой кнопки мыши.

Листинг 18.14. Скетч для управления курсором мыши в исполняющейся на компьютере программе

```
/*
* Скетч Mouse Emulation
* Использует библиотеку Mouse для эмулирования USB-мыши
*/

#include "Mouse.h"

const int buttonPin = 2; // Контакт для подключения кнопки, играющей
                         // роль левой кнопки мыши
const int potXPin = A4;  // Номера аналоговых контактов для подключения потенциометров
const int potYPin = A5;

int last_x = 0;
int last_y = 0;

void setup()
{
   Serial.begin(9600);
   pinMode(buttonPin, INPUT_PULLUP);

   // Получаем начальные позиции потенциометров.
   // Допустимый диапазон значений – от -127 до 127
   last_x = (512 - (int) analogRead(potXPin)) / 4;
   last_y = (512 - (int) analogRead(potYPin)) / 4;
   Mouse.begin();
}

void loop()
{
   // Получаем текущие положения потенциометров
   int x = (512 - (int) analogRead(potXPin)) / 4;
   int y = (512 - (int) analogRead(potYPin)) / 4;

   Serial.print("last_x: "); Serial.println(last_x);
   Serial.print("last_y: "); Serial.println(last_y);
   Serial.print("x: "); Serial.println(x);
   Serial.print("y: "); Serial.println(y);

   // Вычисляем пройденное расстояние на основании состояния
   // потенциометров
   int xDistance = last_x - x;
   int yDistance = last_y - y;

   // Обновляем последние известные положения потециометров
   last_x = x;
   last_y = y;
```

```
  // Если перемещение по оси X или Y больше 3, перемещаем:
  if (abs(xDistance) > 3 || abs(yDistance) > 3)
  {
    Serial.print("x move: "); Serial.println(xDistance);
    Serial.print("y move: "); Serial.println(yDistance);
    Mouse.move(xDistance, yDistance, 0);
  }

  // Если нажата "кнопка" мыши
  if (digitalRead(buttonPin) == LOW)
  {
    if (!Mouse.isPressed(MOUSE_LEFT))
    {
      Mouse.press(MOUSE_LEFT); // Click
    }
  }
  else
  {
    if (Mouse.isPressed(MOUSE_LEFT))
    {
      Mouse.release(MOUSE_LEFT); // Кнопка отпущена
    }
  }
  Serial.println();
  delay(10);
}
```

Обсуждение работы решения и возможных проблем

Этот метод управления исполняющимися на компьютере приложениями легко реализуется и должен работать с любой операционной системой, поддерживающей использование USB-мыши. Если нужно инвертировать перемещение курсора по оси *x* или *y*, это можно сделать, изменив знак переменных `xDistance` и/или `yDistance`, как показано в следующей строке кода:

```
Mouse.move(-xDistance, -yDistance, 0);
```

При исполнении этого скетча возможно возникновение ситуации, в которой теряется возможность управлять курсором мыши с помощью обычной мыши. В таком случае будет затруднительно загрузить в плату новый скетч. На большинстве плат с микроконтроллером архитектуры ARM эта проблема решается двойным нажатием кнопки сброса платы, что переводит ее в режим загрузчика.

Дополнительная информация

Более подробная информация по библиотеке Mouse приведена на соответствующей странице справки веб-сайта Arduino (**https://oreil.ly/eWRLE**).

Библиотека MIDIUSB (**https://oreil.ly/T9P2w**) позволяет платам Arduino с микроконтроллером ATmega32u4 и платам с микроконтроллером архитектуры ARM эмулировать USB-устройство MIDI.

Библиотека ArduinoJoystick (**https://oreil.ly/r6szG**) позволяет платам Arduino с микроконтроллером ATmega32u4 эмулировать джойстик.

Встроенная в среду Arduino IDE библиотека Keyboard (**https://oreil.ly/EN4Mo**) позволяет платам Arduino с микроконтроллером ATmega32u4 и платам с микроконтроллером архитектуры ARM эмулировать USB-клавиатуру.

Библиотека HID (**https://github.com/NicoHood/HID**) позволяет плате Arduino эмулировать различные USB-устройства.

ПРИЛОЖЕНИЕ 1

Электронные компоненты

Для тех, кто только начинает работать с электронными схемами, удобнее приобрести специально подготовленный набор для начинающих, который содержит базовые электронные компоненты, используемые во многих решениях этой книги. Такие наборы обычно включают наиболее распространенные резисторы, конденсаторы, транзисторы, диоды, светодиоды и переключатели.

Далее приводятся ссылки на несколько популярных наборов для начинающих[1]:

- набор Arduino Starter Kit (**https://oreil.ly/oy3YA**);
- набор Getting Started with Arduino Kit (**https://oreil.ly/zhPCi**);
- набор SparkFun Tinker Kit (**https://oreil.ly/akaI4**);
- набор Adafruit MetroX Classic Kit (**https://oreil.ly/jSRPU**);
- набор ARDX: The starter kit for Arduino (**https://oreil.ly/rwakd**).

Требуемые для проектов электронные компоненты можно также приобретать и по отдельности. В следующих разделах приводится краткий обзор наиболее часто используемых электронных компонентов (рис. П1.1).

Конденсатор

Конденсаторы сохраняют электрический заряд в течение короткого времени и используются в цифровых схемах для отфильтровывания (сглаживания) провалов и выбросов напряжения в электрических сигналах. Наиболее распространены неполяризованные керамические конденсаторы — например, дисковый конденсатор 100 нФ, применяемый для сглаживания всплесков напряжения на сигнальных линиях. Электролитические конденсаторы обычно могут сохранять больший заряд, чем керамические конденсаторы, и используются в сильноточных схемах — например, в источниках питания и схемах управления электродвигателями. Электролитические конденсаторы обычно поляризованы, и их отрицательный вывод (обозначенный знаком «минус») нужно подключать на линию «земли» (или к точке с более низким напряжением, чем в точке, к которой подключается положительный

[1] Ссылка **https://bhv.ru/product-category/nabory-dlya-mejkerov/** приведет вас на страницу веб-сайта издательства «БХВ», где предлагаются наборы электронных компонентов как для юных программистов и конструкторов, так и для радиолюбителей всех возрастов — от старшеклассников до пенсионеров.

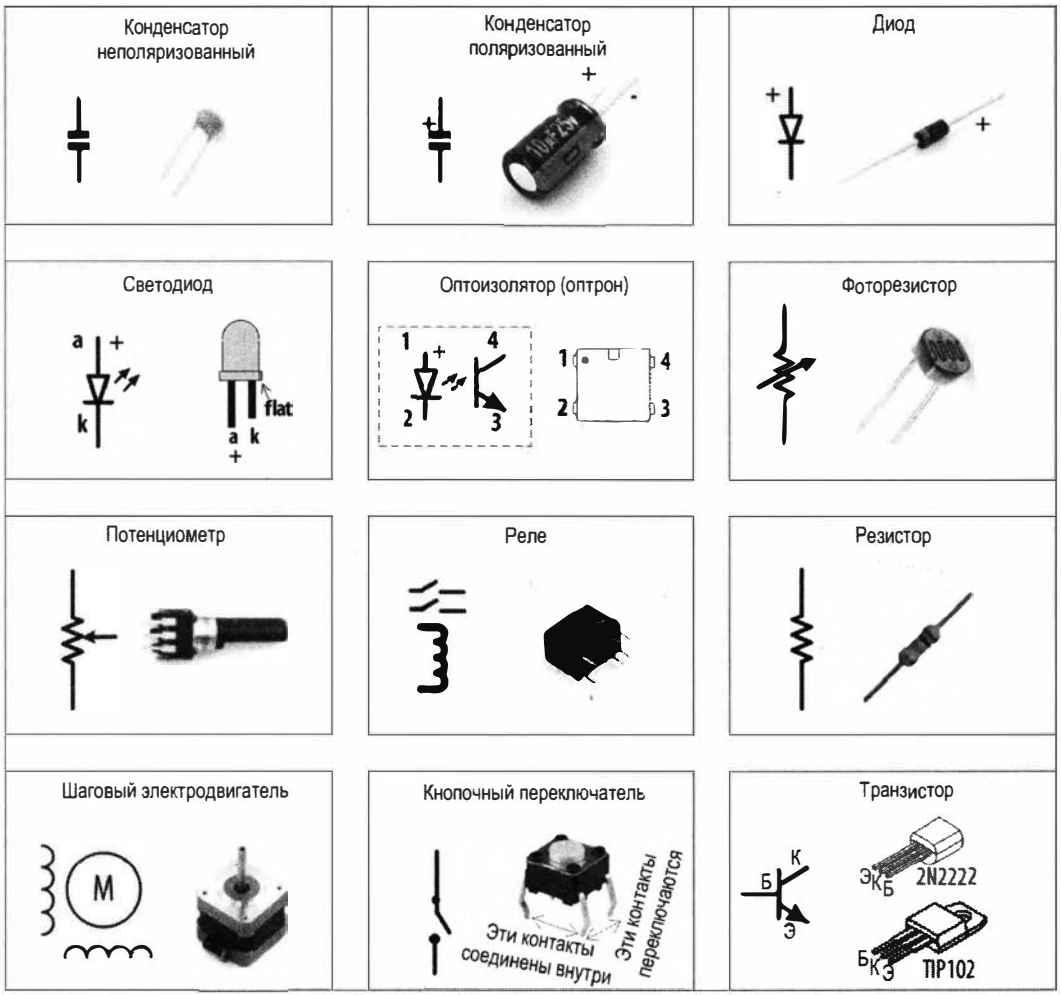


Рис. П1.1. Внешний вид и схематическое представление распространенных электронных компонентов

вывод конденсатора). *Глава 8* содержит несколько примеров использования конденсаторов в схемах управления электродвигателями.

Диод

Диоды разрешают протекание тока в одном направлении и блокируют в другом. Катод (отрицательный вывод) большинства диодов обозначается на нем кольцевой полоской, как показано на рис. П1.1.

Такие диоды, как 1N4148, можно использовать для работы со слаботочными сигналами — например, с логическими уровнями на цифровых контактах платы Arduino. А диод 1N4001 можно задействовать в схемах и для более сильных токов — вплоть до 1 ампера.

Интегральные схемы

Интегральные схемы (или просто микросхемы) содержат электронные компоненты, упакованные в общий корпус. Микросхемы могут быть сложными — такими как микроконтроллер платы Arduino, который содержит тысячи транзисторов, или такими простыми, как оптоизолятор (оптрон), используемый в решениях *главы 10*, который содержит лишь два полупроводниковых элемента. Некоторые микросхемы, включая микроконтроллер платы Arduino, могут быть повреждены разрядом статического электричества, и при обращении с ними следует учитывать это обстоятельство.

Цифровая клавиатура

Цифровая клавиатура представляет собой матрицу переключателей, используемую для ввода цифровых символов. Цифровые клавиатуры рассматриваются более подробно в *главе 5*.

Светодиоды

Светодиод представляет собой диод, который излучает свет при протекании через него тока. Поскольку светодиоды являются диодами, то они пропускают ток только в одном направлении. Светодиоды описаны более подробно в *главе 7*.

Электродвигатели постоянного тока

Электродвигатели преобразовывают электрическую энергию в физическое движение. Скорость вращения вала большинства небольших электродвигателей постоянного тока прямо пропорциональна подаваемому на них напряжению питания. Направление вращения вала таких электродвигателей можно изменить на обратное, поменяв местами линии питания. Большинство электродвигателей потребляют больший ток, чем могут предоставить контакты платы Arduino, поэтому для управления электродвигателями требуется дополнительный компонент — обычно это транзистор. Тема управления электродвигателями рассматривается более подробно в *главе 8*.

Оптрон

Оптроны (оптоизоляторы) обеспечивают электрическую развязку между устройствами. Это дает возможность устройствам с разными напряжениями питания безопасно работать вместе. Использование оптронов более подробно описано в *главе 10*.

Фоторезисторы

Фоторезисторы представляют собой переменные резисторы, чье сопротивление меняется в зависимости от уровня их освещенности. Фоторезисторы рассматриваются более подробно в *главе 6*.

Пьезоэлектрический зуммер

Зуммер — это небольшое керамическое устройство, которое издает звук при подаче на него импульсного сигнала. Пьезозуммер представляет собой поляризованный компонент, положительный вывод которого может обозначаться проводом красного цвета, а отрицательный — черного. Использование пьезозуммера рассматривается более подробно в *главе 9*.

Потенциометр

Потенциометр — это переменный резистор. Два крайних вывода потенциометра подключены к постоянному сопротивлению. По этому постоянному сопротивлению от одного его края к другому перемещается ползунок, к которому подключен средний вывод, в результате чего меняется сопротивление между средним выводом и крайними выводами. Потенциометры подробно рассматриваются в *главе 5*.

Реле

Реле представляет собой электронный переключатель, контакты которого замыкаются и размыкаются якорем, который перемещается под воздействием напряжения, подаваемого на рабочую обмотку реле. Рабочая обмотка реле электрически изолирована от управляемых контактов. Обмотки большинства реле потребляют намного больше тока, чем может предоставить контакт платы Arduino, поэтому для управления ими нужно использовать транзистор. Реле рассматриваются более подробно в *главе 8*.

Резистор

Резисторы оказывают сопротивление протеканию электрического тока. Величина тока, протекающего через резистор под воздействием прилагаемого напряжения, обратно пропорциональна величине сопротивления резистора (закон Ома). Номинальное сопротивление резистора наиболее часто обозначается разноцветными круговыми полосками на его корпусе. В *главе 7* приводится информация по выбору резисторов для использования со светодиодами.

Соленоид

Соленоид состоит из катушки с обмоткой и металлического сердечника (якоря) внутри катушки, который перемещается под воздействием магнитного поля, созда-

ваемого при протекании тока через обмотку. Соленоид рассматривается более подробно в *главе 8*.

Динамик

Динамик создает звук за счет колебаний диффузора, создающего звуковые волны. Диффузор колеблется под воздействием электромагнитного поля, создаваемого в прикрепленной к нему катушке с обмоткой электрическим сигналом звуковой частоты. Динамики рассматриваются более подробно в *главе 9*.

Шаговый электродвигатель

Шаговый электродвигатель вращает вал на определенное количество градусов в ответ на импульсы управления. Использование шаговых электродвигателей рассматривается более подробно в *главе 8*.

Переключатель

Переключатель замыкает и размыкает электрическую цепь. Во многих решениях этой книги используются кнопочные переключатели, которые также называются *тактильными кнопками* или просто *кнопками*. Тактильная кнопка имеет две пары контактов, которые замыкаются при ее нажатии. Пары контактов соединены вместе внутри корпуса кнопки, что позволяет замыкать/размыкать две линии одновременно. Кнопочные переключатели, замыкающие цепь при нажатии, называются *нормально разомкнутыми*. Переключатели рассматриваются более подробно в *главе 5*.

Транзистор

Транзисторы применяются в цифровых схемах для управления сильными токами и/или высокими напряжениями. В аналоговых схемах транзисторы используются для усиления сигналов. Небольшой ток, протекающий через базу транзистора, вызывает намного больший ток, протекающий в цепи «коллектор — эмиттер».

Для приложений с токами до 0,5 ампера (500 мА) широко используется транзистор 2N2222. Для токов до 5 ампер можно использовать транзистор TIP120.

Примеры использования транзисторов для управления светодиодами и электродвигателями приводятся в *главах 7* и *8*.

Дополнительная информация

Более углубленно базовая электроника рассматривается в книгах, список которых приводится в *разд. «Что не вошло в эту книгу?» предисловия*.

Работа с принципиальными схемами и справочными листками

Принципиальные схемы

Стандартным способом описания компонентов и их соединений в электронной схеме является ее *принципиальная схема*. Для представления электронных компонентов используются символьные значки, а линиями показываются соединения между компонентами.

На рис. П1.1 (см. *приложение 1*) показаны наиболее часто употребляемые электронные компоненты и соответствующие им символьные обозначения, применяемые в принципиальных схемах. На рис. П2.1 приводится принципиальная схема из *разд. 8.8*, содержащая символы электронных компонентов, используемые в типичной электронной схеме.

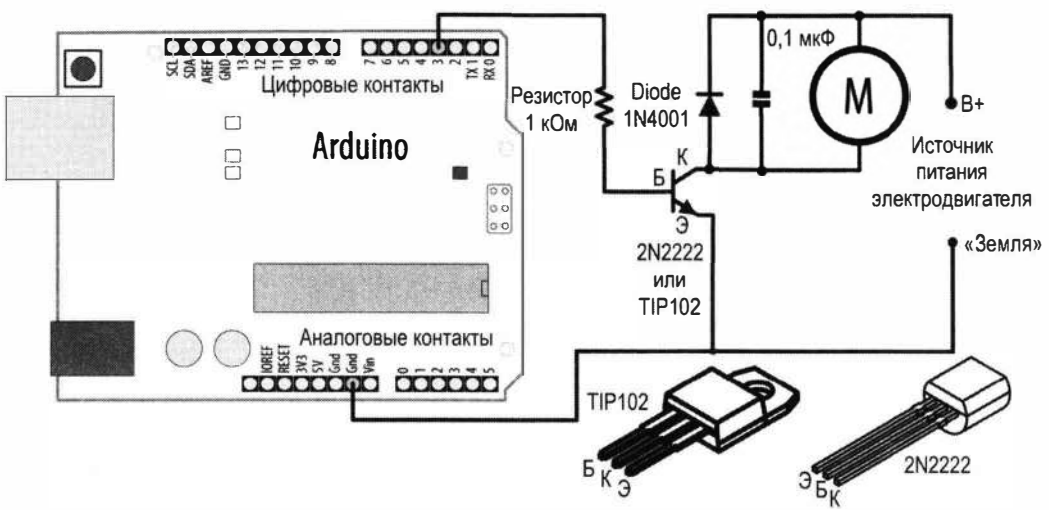


Рис. П2.1. Типичная принципиальная схема

Принципиальная схема показывает компоненты схемы и их соединения, но не является рисунком или чертежом собственно физического размещения и соединений компонентов. Хотя на первых порах в чертеже или рисунке (монтажной схеме) физических соединений простых электронных схем вам может быть разобраться

легче, чем в принципиальной схеме, в более сложных электронных схемах гораздо труднее понять, куда подключен каждый провод. Принципиальные схемы можно сравнить, например, с географическими картами. Они следуют установленным правилам, которые помогают пользователю, знающему обозначения и стиль представления используемых символов, сориентироваться в организации схемы. Например, входы обычно расположены слева, а выходы — справа, шина положительного питания обычно отображается вверху, а отрицательного («земля») — внизу и т. п. Выводы неполяризованных компонентов, таких как резисторы и конденсаторы, можно подключать любым способом. Выводы транзисторов, диодов и микросхем поляризованы, поэтому важно определить назначение каждого вывода и подключить его в соответствии с принципиальной схемой.

На рис. П2.2 показано, как выглядит соединение компонентов при сборке схемы из рис. П2.1 на беспаечной макетной плате. Этот рисунок создан в программе Fritzing (**http://fritzing.org**), которая позволяет создавать рисунки электронных схем (монтажные схемы).

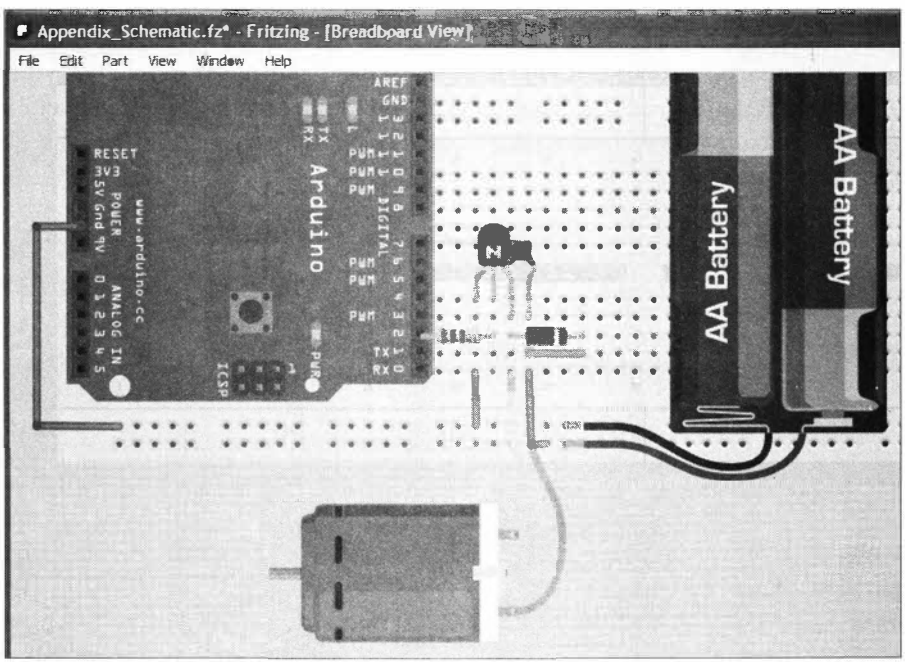


Рис. П2.2. Рисунок физической организации схемы, показанной на рис. П2.1

Собрать рабочую электронную схему на макетной плате по ее принципиальной схеме очень просто, если разбить эту задачу на несколько отдельных шагов. На рис. П2.3 показано, как каждый шаг сборки схемы на макетной плате связан с принципиальной схемой (здесь собирается схема из решения в *разд. 1.6*).

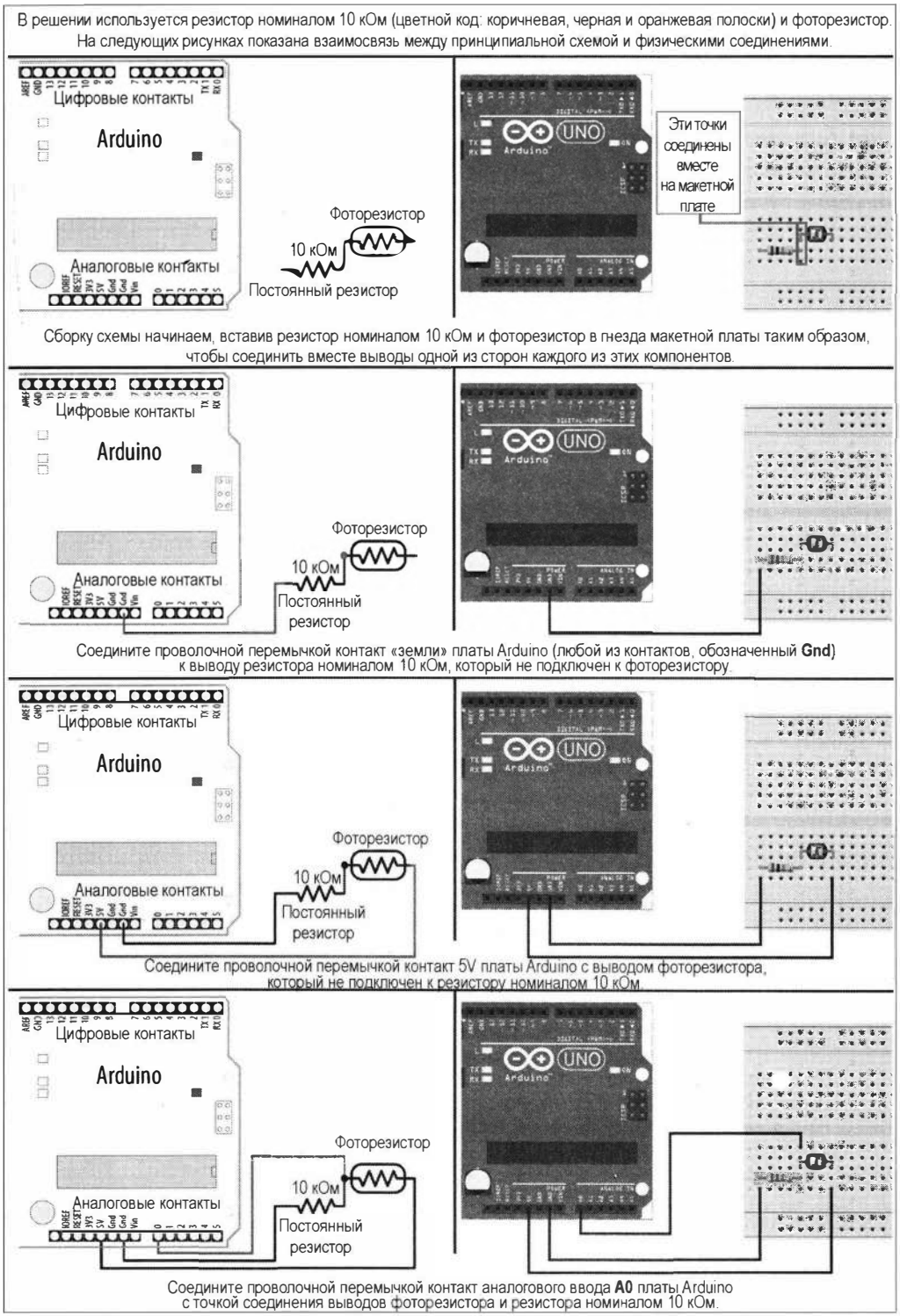


Рис. П2.3. Перенос принципиальной схемы на макетную плату

Справочные листки

Справочные листки (datasheets) издаются производителями электронных компонентов, чтобы предоставить пользователям основные технические характеристики своих устройств. Справочные листки содержат наиболее точную информацию о технических характеристиках устройства и способах его применения — например, минимальное рабочее напряжение, при котором устройство может надежно работать, и максимальное напряжение, которое устройство может гарантированно выдерживать. Справочные листки также включают информацию о назначении каждого контакта устройства и советы, как использовать устройство.

В случае более сложных устройств — таких как, например, жидкокристаллические дисплеи, справочный листок содержит информацию о том, как инициализировать устройство и взаимодействовать с ним. А справочные листки для очень сложных устройств — таких как микроконтроллер платы Arduino, на самом деле представляют собой целые книги на несколько сотен страниц, объясняющие все возможности устройства.

Справочные листки на устройства составляются их инженерами-разработчиками и обычно содержат намного больше информации, чем необходимо для того, чтобы собрать из этих устройств рабочую схему проекта Arduino. Не пугайтесь большого объема технической информации — обычно требуемую вам информацию можно найти на первых нескольких страницах справочного листка. Эти страницы обычно демонстрируют символ устройства, используемый для его представления на принципиальных схемах, с обозначениями назначения всех его выводов, а также содержат общее описание устройства (или семейства устройств) и типов задач, для решения которых оно может применяться. Далее обычно следует таблица электрических характеристик устройства, в которой можно найти информацию о максимальном рабочем напряжении и токе, чтобы проверить, соответствуют ли они требуемым. Рабочее напряжение устройств, подключаемых напрямую к контактам стандартной платы Arduino, должно быть равно 5 вольт. Кроме этого, потребляемый такими устройствами ток не должен превышать 40 мА.

Рабочее напряжение некоторых устройств составляет 3,3 В. Такие устройства могут быть повреждены при прямом подключении их к контактам платы Arduino с рабочим напряжением 5 В. Поэтому их следует использовать с платами с рабочим напряжением 3,3 В (например, платами серии MKR, платами с микроконтроллером архитектуры ARM Cortex-M0, такими как Arduino Zero и др.). Впрочем, для подключения устройств с рабочим напряжением 3,3 В к платам Arduino с рабочим напряжением 5 В можно использовать преобразователь логических уровней — например, устройство с артикулом BOB-08745 компании SparkFun. Более подробная информация на тему преобразования уровней приводится в документе по использованию **https://oreil.ly/waiXi**.

Справочные листки и выбор транзисторов для использования в схемах Arduino

Контакты Arduino могут работать с токами, не превышающими 40 миллиампер, что равно всего лишь $^{1}/_{25}$ ампера. Токовые возможности контактов других плат могут

быть еще более низкими. Например, контакты платы Arduino Uno WiFi Rev2 могут обеспечить ток величиной 20 мА, а платы Arduino Zero — всего лишь 7 мА. Но проблему управления более сильными токами можно решить с помощью транзистора. Здесь мы рассмотрим, как выбрать подходящий транзистор для конкретного приложения.

В проектах Arduino наиболее часто используются биполярные транзисторы. По направлению протекания тока различаются два типа таких транзисторов: NPN и PNP. В проектах Arduino (и в решениях этой книги) чаще всего применяются транзисторы типа NPN: для работы с токами до 0,5 ампера (500 мА) или около этого — транзистор 2N2222, а для более сильных токов вплоть до 5 ампер — транзистор TIP120.

В схеме на рис. П2.1 показан пример использования транзистора для управления электродвигателем платой Arduino. В *главе 8* приводится несколько других решений, в которых задействованы транзисторы.

Справочные листки транзисторов обычно загружены информацией для инженеров-конструкторов, но большая часть этой информации не нужна для выбора транзисторов, используемвх в проектах с Arduino. В табл. П2.1 приведены наиболее важные параметры, которые следует учитывать при выборе транзистора для конкретного приложения (показаны значения для типичного транзистора общего назначения). В результате производственной погрешности одинаковые транзисторы разных партий изготовления могут иметь различающиеся характеристики, поэтому в справочных листках обычно указываются минимальные, типичные и максимальные значения параметров, точные значения которых могут варьироваться от одного транзистора к другому.

***Таблица П2.1**. Пример основных параметров транзисторов, представляемых в справочном листке*

Абсолютное максимальное значение				
Параметр	**Обозначение**	**Значение**	**Единицы**	**Описание**
Напряжение «коллектор — эмиттер»	$U_{КЭ}$	40	Вольты	Максимальное напряжение между коллектором и эмиттером
Ток коллектора	$I_К$	600	мА или А	Максимальный ток, с которым может работать транзистор
Электрические характеристики по напряжениям постоянного тока				
Усиление по постоянному току	$I_К$	90 при 10 мА		Усиление при токе величиной 10 мА
		50 при 500 мА		Усиление при токе величиной 500 мА
Напряжение насыщения «коллектор — эмиттер»	$U_{КЭ}$ (насыщ.)	0,3 при 100 mA	Вольты	Падение напряжения между коллектором и эмиттером при разных токах
		1,0 при 500 мА	Вольты	

Итак, при выборе транзистора следует рассмотреть следующие параметры:

- **напряжение «коллектор — эмиттер»** — убедитесь, что транзистор способен работать с более высоким напряжением, чем напряжение питания схемы, которой он должен управлять. Более высокое значение этой характеристики не будет создавать никаких проблем;
- **ток коллектора** — максимальный ток, с которым может работать транзистор. Рекомендуется, чтобы значение этой характеристики было как минимум на 25% больше, чем требуемое значение;
- **усиление по постоянному току** — этот параметр определяет величину тока, который должен протекать через базу транзистора, чтобы переключать выходной ток. Величину этого тока можно получить, разделив выходной ток (максимальный ток, протекающий через переключаемую транзистором нагрузку) на усиление по постоянному току. Значение сопротивления резистора, через который контакт платы Arduino подключается к базе транзистора, вычисляется с помощью закона Ома: $R = U/I$. Например, если требуемый ток коллектора составляет 1 ампер, а усиление равно 100, то через базу должен протекать ток величиной минимум 0,01 А (10 мА). Для платы Arduino с питанием 5 В это: 5В / 0,01 А = 500 Ом. Поскольку 500 Ом не является стандартным номиналом резистора, возьмем ближайший стандартный номинал 470 Ом;
- **напряжение насыщения «коллектор — эмиттер»** — это напряжение на коллекторе при полностью открытом (проводящем) транзисторе. Хотя обычно это значение меньше чем 1 вольт, оно может иметь важность при вычислении значения сопротивления последовательного резистора для светодиодов или для управления сильноточными устройствами.

ПРИЛОЖЕНИЕ 3

Сборка схем

Существуют разные способы монтажа компонентов при сборке схем. Но для сборки прототипов схем наиболее часто используется беспаечная макетная плата. Для правильной работы собранных схем необходимо обеспечить качественный источник питания, не допускающий колебаний тока и напряжения. В этом приложении мы рассмотрим базовые методы монтажа схем на беспаечных макетных платах, а также дадим несколько советов по работе с источниками питания.

Работа с беспаечной макетной платой

Беспаечная макетная плата позволяет быстро создавать прототипы схем, не требуя прибегать к пайке для соединения компонентов. На рис. П3.1 показано, как выглядит беспаечная макетная плата.

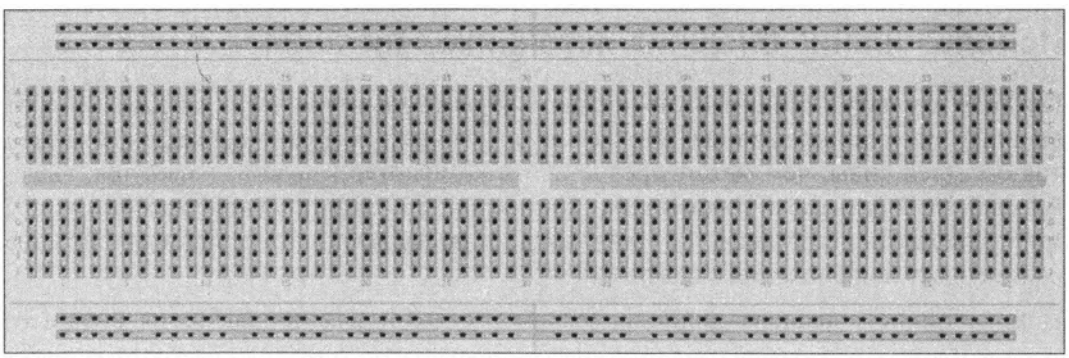


Рис. П3.1. Беспаечная макетная плата для быстрого создания прототипов схем

На рынке предлагаются беспаечные макетные платы разных типов и конфигураций. Самые простые из них представляют собой просто пластмассовый блок с матрицей отверстий. Отверстия (или гнезда) в столбцах электрически соединены между собой металлическими полосками, расположенными в желобках внутри блока. Если вставить выводы двух разных компонентов в два разных гнезда одного столбца гнезд, они попадут в пружинящий металлический контакт и зажмутся в нем, в результате чего эти два компонента будут электрически соединены. Посередине макетной платы вдоль ее длинной стороны проходит выемка, разделяющая столбцы гнезд одной стороны от другой. Это означает, что столбцы гнезд на противоположных сторонах выемки электрически не связаны между собой и компоненты, встав-

ленные в гнезда одного и того же столбца, но по разным сторонам выемки, не будут соединены электрически.

Более продвинутые макетные платы оснащены шинами питания в виде одного или двух рядов гнезд вдоль длинного края платы, на некотором расстоянии от столбцов гнезд для монтажа компонентов. Эти гнезда соединены между собой такими же металлическими полосками, как и гнезда вертикальных столбцов. На эти гнезда можно подключить напряжение питания от источника питания, а уже с них подавать питание на собираемую схему. Если макетная плата имеет по одному ряду гнезд питания на каждой из сторон, на один из них подается плюс, а на другой — минус («земля») питания. Если же с каждой стороны платы имеется по паре шин питания (как на рис. П3.1), то на одну линию пары подается плюс, а на другую — минус питания. Таким образом можно организовать питание с двумя разными напряжениями, используя одну из пар шин для, например, напряжения 5 В, а другую — для напряжения 3,3 В. Благодаря наличию таких шин значительно упрощается подача питания на разрабатываемую схему, позволяя подавать питание на разные компоненты или в разные точки схемы.

Макетные платы хороши для монтажа прототипов схем, однако они имеют свои недостатки и ограничения. Поскольку электрические соединения в них создаются за счет трения, они не такие надежные, как паяные. Поэтому, если у вас периодически возникают проблемы с работой схемы, это может быть вызвано плохим контактом одного или нескольких компонентов, вставленных в гнезда макетной платы.

Использование внешних источников питания

Питание на плату Arduino можно подавать не из компьютера через кабель USB, а от внешнего источника питания. Это может потребоваться в том случае, когда нужен ток большей силы, чем может предоставить компьютер (максимальный ток, выдаваемый USB-разъемом компьютера, составляет 500 мА, а некоторые USB-хабы могут выдавать всего лишь 100 мА). Или же после загрузки скетча плата должна работать без подключения к компьютеру.

Плата Arduino оснащена цилиндрическим разъемом для подключения внешнего источника питания, которым может быть или адаптер с питанием от электросети, или батарейка.

Излагаемое здесь относится к платам Arduino Uno и Mega. Другие платы Arduino и совместимые платы могут не обладать защитой от напряжения обратной полярности, или могут не переключаться автоматически на внешний источник питания, или могут быть неспособными работать с более высокими напряжениями питания. Поэтому, в случае использования плат иных, чем Arduino Uno или Mega, следует изучить особенности подключения внешнего питания на эти платы, прежде чем делать это, поскольку неправильное подключение внешнего питания может повредить плату.

При использовании сетевого источника питания он должен выдавать выходное напряжение постоянного тока в диапазоне от 7 до 12 вольт. Кроме этого, он должен выдавать как минимум ток требуемой силы (ток большей силы не вызовет пробле-

мы). Сетевые блоки питания, которые непосредственно вставляются в розетку, могут быть стабилизированными или нестабилизированными. Стабилизированный сетевой источник питания оснащен схемой, которая удерживает постоянное напряжение при меняющейся нагрузке. Для питания платы Arduino рекомендуется использовать источники питания именно этого типа. Нестабилизированные источники питания выдают более высокое напряжение при малых нагрузках, которое иногда может оказаться вдвое выше номинального напряжения питания слаботочных устройств, одним из которых является плата Arduino. Напряжение выше 12 вольт может вызвать перегрев стабилизатора питания на плате Arduino, что, в свою очередь, повлечет за собой нестабильную работу платы или даже выведет ее из строя.

Напряжение батарейного источника питания также должно быть в диапазоне от 7 до 12 вольт. Основной характеристикой батарей является их емкость, которая выражается в миллиампер·часах (или ампер·часах) и представляет количество часов, в течение которых батарея может (теоретически) нормально работать, поставляя ток величиной в один миллиампер (или один ампер). При большем или меньшем потребляемом токе количество рабочих часов соответственно сокращается или увеличивается. Например, батарея емкостью 500 мА (типичная щелочная батарейка напряжением 9 В) способна обеспечить нормальную работу платы Arduino, потребляющей ток величиной 25 мА, в течение приблизительно 20 часов (500 мА·часов : 25 мА = 20 часов). Если же проект потребляет ток величиной 50 мА, то количество рабочих часов батареи будет соответственно уменьшено вдвое — до приблизительно 10 часов. Величина потребляемого тока в основном зависит от используемых в проекте устройств — таких как светодиоды, электродвигатели и т. п. Следует иметь в виду, что плата Arduino Uno разрабатывалась с прицелом на легкость использования и надежность в работе, но не оптимизирована для работы в энергосберегающем режиме от батареи. Потребление тока платой можно понизить в большей или меньшей степени, следуя советам, приведенным в *разд. 18.10*.

Положительная линия внешнего источника питания должна подключаться к центральному штырьку цилиндрического разъема внешнего питания платы Arduino. Подключение внешнего питания обратной полярности к плате Arduino Uno или Mega не вызовет повреждения платы, но плата (естественно) не будет работать, пока питание не будет подключено с правильной полярностью. Эти платы автоматически определяют подключение внешнего источника питания и используют его для питания платы. При этом можно одновременно использовать USB-подключение для загрузки кода и взаимодействия с компьютером по последовательному интерфейсу.

Использование развязывающих конденсаторов

Цифровые схемы переключают состояние логических сигналов с очень высокой скоростью, что может вызывать колебания напряжения источника питания, что, в свою очередь, может повлечь за собой нестабильную работу схемы. Для предотвращения такого развития событий следует использовать развязывающие (блоки-

рующие) конденсаторы, чтобы отфильтровывать (сглаживать) такие колебания. Развязывающие конденсаторы подключаются параллельно контактам питания каждой микросхемы разрабатываемой схемы, при этом длина выводов конденсаторов должна быть минимально короткой. Хорошим выбором развязывающего конденсатора станет керамический конденсатор емкостью 0,1 мкФ, но точная емкость не является критической и допускается погрешность величиной ±20 процентов.

Использование демпферных диодов с индуктивными нагрузками

К индуктивным нагрузкам относятся устройства, содержащие обмотки, — включая электродвигатели, соленоиды и реле. При подаче или прекращении подачи питания на обмотку создается всплеск тока, вызывающий соответствующий выброс напряжения. Величина этого напряжения может превышать +5 В и способна повредить чувствительные электронные компоненты, такие как схемы, управляющие контактами платы Arduino. Для борьбы с этим явлением используются демпферные диоды, которые отводят ток на «землю». На рис. П2.1 показан пример использования демпферного диода в цепи питания электродвигателя.

Работа с напряжением электросети

При работе с напряжением электросети первое, чем следует озаботиться, — можно ли вообще избежать работы с ним. Высокое напряжение электросети достаточно опасное, и при несоблюдении правил работы с ним может убить не только разрабатываемую схему, но и самого разработчика. Кроме этого, при неправильно выполненной изоляции оно также может представлять опасность и для пользователей устройства.

Модифицировать соответствующим образом контроллеры устройств, предназначенных для работы с сетевым электричеством, или использовать устройства, специально предназначенные для работы с микроконтроллерами для управления сетевым электричеством, гораздо безопаснее (и часто легче), чем работать напрямую с сетевым электричеством. В *главе 10* показано, как это сделать.

Советы по диагностированию программных неполадок

В процессе разработки и модифицирования кода может случиться, что полученный код по какой-либо причине (эта причина обычно называется *багом*) отказывается работать должным образом. Проблемы с кодом можно грубо разделить на два типа: код, который не компилируется, и код, который компилируется и загружается в плату, но не работает, как от него ожидалось.

Код, который не компилируется

Ситуация, когда при нажатии кнопки **Проверить** или **Загрузка** (см. *главу 1*) код отказывается компилироваться, является весьма распространенной, особенно у начинающих программистов. О наличии проблемы с компилированием говорят сообщения об ошибке, выводимые красным шрифтом в консоли сообщений окна среды Arduino IDE, а конкретная строка кода, создающая проблему, выделяется розовым цветом. Но часто настоящей строкой с проблемой является строка кода, непосредственно предшествующая выделенной. Сообщения об ошибке, выводимые в консоль сообщений окна среды Arduino IDE, создаются программами командной строки, используемыми для компилирования и сборки кода (более подробно о процессе сборки программы рассказано в *разд. 17.1*). Для начинающих программистов эти сообщения об ошибке могут оказаться малопонятными.

Одной из наиболее частых ошибок, допускаемых начинающими программистами на языке Arduino, является пропуск точки с запятой в конце строки кода. В зависимости от следующей за ней строки кода это может вызывать разные сообщения об ошибке. Например, такой фрагмент кода:

```
void loop()
{
   digitalWrite(ledPin, HIGH) // <- BUG: пропущена точка с запятой
   delay(1000);
}
```

вызывает следующее сообщение об ошибке:

```
In function 'void loop()':
error: expected ';' before 'delay
```

Но это сообщение гораздо менее понятное:

```
expected ',' or ';' before 'void'
```

Хотя причина его та же — пропущенная точка с запятой после строки кода объявления константы:

```
const int ledPin = LED_BUILTIN // <- BUG: Пропущенная точка с запятой
                               // после объявления константы
void loop()
```

Совокупность сообщения об ошибке и выделенной строки кода предоставляет хорошую отправную точку для определения области, в которой произошла ошибка.

Другой распространенной ошибкой являются неправильно написанные ключевые слова или названия переменных, в результате чего компилятор их не распознает. К ошибкам написания также относится неправильная *капитализация* (использование заглавных букв): название `LedPin` — это не то же самое, что `ledPin`. Следующий фрагмент кода:

```
const int ledPin = LED_BUILTIN;

digitalWrite(LedPin, HIGH); // <- BUG: Заглавная буква в названии переменной
```

вызывает следующее сообщение об ошибке:

```
note: suggested alternative: 'ledPin':
error: 'LedPin' was not declared in this scope
```

Эта проблема решается точно таким же написанием названий переменных, включая использование заглавных (прописных) и строчных букв, как и в их объявлениях.

При вызове функций необходимо передавать им правильное количество параметров правильного типа (см. *разд. 2.10*). Следующая строка кода:

```
digitalWrite(ledPin); // <- BUG: Упущен второй параметр
```

создает следующее сообщение об ошибке:

```
error: too few arguments to function 'void digitalWrite(uint8_t, uint8_t)'
error: at this point in file
```

Курсор среды Arduino IDE будет указывать на строку кода, содержащую ошибку.

Ошибку также вызывает упущение типа возвращаемого функцией значения. Следующий фрагмент кода:

```
loop() // <- BUG: Для функции loop() не указан тип возвращаемого ею значения
{
}
```

создает следующее сообщение об ошибке:

```
expected constructor, destructor, or type conversion before ';' token
```

Ошибка исправляется указанием типа возвращаемого функцией значения:

```
void loop() // <- Тип возвращаемого значения указывается перед названием функции
{
}
```

Неправильно оформленные комментарии также вызывают ошибки компиляции. Например, пропущенная вторая косая черта:

```
digitalWrite(ledPin, HIGH); / set the LED on (<-BUG: пропущена вторая /)
```

вызывает следующее сообщение об ошибке:

```
error: expected primary-expression before '/' token
```

Рекомендуется последовательно работать над небольшими областями кода, регулярно выполняя операцию проверки/компилирования, чтобы убедиться в правильности синтаксиса созданного кода. При этом загрузку кода в плату выполнять не нужно (просто нажмите кнопку **Проверить** в панели инструментов среды Arduino IDE). Чем раньше вы узнаете о наличии проблемы, тем легче будет обнаружить ее причину и исправить ее и тем меньшее влияние она будет оказывать на другие области кода. Исправить небольшую область кода с одной ошибкой намного легче, чем большую область с несколькими ошибками.

Код компилируется, но не работает, как ожидалось

Компилирование кода без ошибок всегда вызывает чувство удовлетворения от успешно выполненной задачи, но правильный синтаксис не означает, что код будет работать так, как от него ожидается.

Решение таких проблем связано с большими трудностями, чем решение проблем с синтаксисом. Здесь мы находимся в области, в которой происходит взаимодействие программного и аппаратного обеспечения. Для правильного решения проблем важно отделить аппаратные проблемы от программных. Внимательно проверьте работу аппаратного обеспечения (см. *приложение 5*), чтобы убедиться, что все функционирует должным образом.

Диагностирование взаимосвязанных аппаратно-программных проблем

Причиной некоторых проблем являются не строго программные или аппаратные ошибки, а взаимодействие их в этих двух областях.

Наиболее распространенной ошибкой такого типа является подача сигнала на один контакт, а считывание его с другого контакта. По отдельности все правильно, как в аппаратном, так и в программном отношении, но в совокупности код не работает, как ожидается. Эта проблема решается или аппаратно, или программно. Аппаратное решение состоит в подаче сигнала на контакт, используемый в программе, а программное — в использовании контакта, на который подается сигнал.

Если вы уверены, что электронная схема собрана правильно и работает должным образом, на первом шаге поиска причины проблемы в скетче внимательно проверьте используемую в нем логику решения задачи. Уделите немного времени внимательному осмыслению того, что именно происходит в написанном вами коде, — это обычно более быстрый и продуктивный способ решить проблему, чем сразу же приступать к отладке скетча с добавлением в него отладочного кода. Логическую ошибку в только что написанном коде часто бывает весьма трудно обнаружить. В таких случаях полезно отойти на некоторое время от компьютера. Это не только позволит отдохнуть вашим глазам, но также придаст новых сил вашим способностям диагностирования. По возвращении вы станете смотреть на код новым взгля-

дом, и, как часто бывает в таких случаях, причина ошибки, которую раньше вы не могли разглядеть, сама бросится вам в глаза.

Если этот подход не даст положительных результатов, переходите к следующему методу: откройте монитор порта и наблюдайте изменения значений в скетче в процессе его исполнения, а также проверьте выполняются ли условные переходы так, как они должны выполняться. В *главе 4* подробно объясняется, как использовать операторы `print()` языка Arduino для вывода значений в окно монитора порта.

Для отладки проблемного кода нужно знать, что в действительности происходит в процессе его исполнения. Вставив в скетч операторы `Serial.print()`, вы сможете увидеть, какие части кода исполняются при определенных условиях и какие значения переменных они создают. Это врéменные операторы, поэтому после определения причины проблемы и ее устранения их следует удалить. В листинге П4.1 приводится скетч на основе скетча решения из *разд. 5.6*, который считывает значение с аналогового контакта. Скетч должен изменять частоту мигания светодиода на основе значения потенциометра (подробное объяснение схемы и скетча приводится в *разд. «Обсуждение работы решения и возможных проблем» разд. 5.6*). Если решение не работает должным образом, то проверить правильность работы ее программной части, можно, отображая в окне монитора порта считываемые с контакта значения с помощью оператора `Serial.print()`.

Листинг П4.1. Использование оператора `Serial.print()` для отладки скетча

```
const int potPin = A0;
const int ledPin = LED_BUILTIN;
int val = 0;

void setup()
{
   Serial.begin(9600); // <- Добавьте эту строку, чтобы инициализировать объект Serial
   pinMode(ledPin, OUTPUT);
}

void loop()
{
   val = analogRead(potPin); // Считываем напряжение на контакте потенциометра
   Serial.println(val);      // <- Добавьте эту строку для вывода
                             // считываемых значений в окно монитора порта
   digitalWrite(ledPin, HIGH);
   delay(val);
   digitalWrite(ledPin, LOW);
   delay(val);
}
```

Если при вращении вала потенциометра выводимые в окне монитора порта значения не меняются в диапазоне от 0 до 1023, возможно, что это вызвано аппаратной проблемой — например, неисправным потенциометром или его неправильным подключением. Если значения меняются, а светодиод не мигает, тогда, возможно, светодиод подключен неправильно.

ПРИЛОЖЕНИЕ 5

Советы по диагностированию аппаратных неполадок

Аппаратные проблемы способны вызвать более быстрые и серьезные последствия, чем программные, поскольку при неправильном монтаже электронные компоненты схемы могут быть повреждены. Самый важный совет тут — это всегда отключать питание, прежде чем подключать или отключать компоненты, и проверять и перепроверять правильность монтажа, прежде чем подключать питание, — подача питания является не первым, а последним шагом в тестировании схемы

При сборке и/или модификации схем, подключенных к плате Arduino, всегда физически отсоединяйте плату от источника питания.

Монтаж сложных схем следует выполнять небольшими шагами. Часто сложные схемы состоят из нескольких отдельных схемных элементов, каждый из которых подключается к своему контакту платы Arduino. В таких случаях следует собрать один схемный элемент, протестировать его, затем другой элемент и т. д. При наличии возможности каждый элемент следует тестировать, используя заведомо рабочий скетч, — например, один из скетчей примеров среды Arduino IDE или примеров, предлагаемых на площадке Arduino Playground (**https://playground.arduino.cc**). Как правило, тестирование каждого элемента схемы по отдельности позволяет запустить сложный проект в работу намного быстрее.

Для использования некоторых рассматриваемых здесь методов требуется наличие мультиметра (подойдет любой недорогой цифровой измерительный прибор, способный измерять напряжение, ток и сопротивление).

Самый эффективный способ тестирования собираемой схемы — внимательное исследование ее монтажа, убеждаясь, что он соответствует принципиальной схеме. Особенное внимание следует уделять правильности подключения линий питания, избегая создания коротких замыканий, когда плюс питания случайно подключается к минусу, или когда выводы компонентов соприкасаются в местах, где они не должны соприкасаться. Если вы не уверены, какой ток будет потреблять устройство, подключенное к контакту Arduino, проверьте это с помощью мультиметра, прежде чем подключать его. Если устройство потребляет больше чем 40 мА (20 мА — при подключении к плате Arduino Uno WiFi Rev2 или Nano Every, или 7 мА — при подключении к большинству плат с микроконтроллером архитектуры ARM), подключение его к контакту Arduino может вывести из строя схему контакта (подробно об использовании мультиметра рассказывается в статье «Getting Started with Your Multimeter», доступной по ссылке: **https://oreil.ly/lUVo9**).

Исправность выходных устройств (светодиодов или электродвигателей) можно проверить, подключив их к источнику питания вместо контакта платы Arduino. Если при подаче на него питания устройство не работает должным образом, оно может быть неисправно или неправильно подключено.

Если же устройство работает при подключении к источнику питания, но не работает при подключении к плате Arduino и исполнении кода, проблемой может быть неисправная схема контакта или неправильный код.

Чтобы проверить исправность схемы контакта платы Arduino, подключите к нему через последовательный резистор светодиод (см. *главу 7*) или мультиметр для измерения напряжения и загрузите в плату скетч Blink, модифицированный для управления этим контактом. Если светодиод не мигает или если напряжение не переключается между 0 и 5 В (или 3,3 В на платах с напряжением питания 3,3 В), то, скорее всего, схема микроконтроллера для управления этим контактом вышла из строя.

Будьте внимательны, чтобы плюс питания не мог случайно закоротиться на «землю». В таком случае плата, на которую питание подается с компьютера через USB-подключение, прекратит функционировать, поскольку сработает самовосстанавливающийся предохранитель платы, не разрешающий потребление чрезмерно большого тока из порта USB компьютера. При превышении определенной силы потребляемого платой тока предохранитель срабатывает и отключает подачу питания на плату. Возвратить его в рабочее состояние можно, отключив плату от порта USB компьютера (может также потребоваться и перезагрузить компьютер). Но прежде чем снова подключать плату к порту USB компьютера, проверьте все подключенные к ней схемы, чтобы найти и исправить вызывающую короткое замыкание неисправность. Если этого не сделать, то при повторном подключении предохранитель снова сработает.

Ничего не помогло?

Может случиться, что вы перепробовали все решения, которые могли вам прийти в голову, а проект все равно отказывается работать должным образом. В таком случае можно обратиться за помощью к кому-либо, кто работает с платами Arduino или другими подобными платами, если у вас на примете есть такой специалист. Но если у вас таких знакомых нет, тогда за помощью нужно идти в Интернет в общем, и на форум Arduino в частности. Здесь пользователи с опытом любого уровня могут задавать вопросы и делиться своими знаниями. Начните с поиска информации, относящейся к вашему проекту, используя строку поиска форума (она находится в правом верхнем углу страницы). Другим местом для поиска ответов на свои вопросы является уже упомянутая ранее площадка Arduino Playground, на которой пользователи делятся своей информаций о платформе Arduino.

Если поиск не даст требуемой вам информации, можно задать вопрос на форуме Arduino. Пользователи форума очень активные, и если ясно сформулировать вопрос, то, скорее всего, вам быстро на него ответят.

Чтобы задать вопрос должным образом, определите, в каком разделе форума его нужно разместить, и выберите для него название, отражающее конкретную проблему, которую вы пытаетесь решить. Размещайте вопрос только в одном месте, т. к. большинство пользователей форума, склонные дать ответ на него, проверяют все разделы с новыми сообщениями, и несколько одинаковых сообщений в разных разделах могут вызвать у них раздражение, в результате чего они могут отказаться помочь вам.

Подробно опишите свою проблему и укажите шаги, предпринятые вами для ее решения. Будет предпочтительнее описать то, что реально происходит, а не то, что, как вам кажется, происходит. Приложите к вопросу весь соответствующий код, но при этом старайтесь создать краткий скетч, который не содержит кода, который не относится к вашей проблеме. Если проблема связана с устройством или компонентом, не входящим в состав платы Arduino, также предоставьте ссылку на его справочный листок. В случае сложного монтажа приложите схему или фотографию, по которой можно ясно видеть, каким образом выполнены все соединения.

ПРИЛОЖЕНИЕ 6

Цифровые и аналоговые контакты

В табл. П6.1 и П6.2 приводятся списки цифровых и аналоговых контактов для плат Arduino Uno и Mega и краткое описание их назначения.

В столбце **Порт** приведен список физических портов для соответствующих контактов (подробно установка уровня контакта посредством записи в порт описана в *разд. 18.11*). В таблице имеются следующие обозначения:

- USART RX — контакт приема аппаратного последовательного интерфейса;
- USART RX — контакт передачи аппаратного последовательного интерфейса;
- Ext Int — внешнее прерывание (с соответствующим номером);
- PWM TnA/B — выходной ШИМ-сигнал (функция `analogWrite()`) на таймере *n*;
- MISO, MOSI, SCK и SS — сигналы управления интерфейса SPI;
- SDA и SCL — сигналы управления интерфейса I^2C.

Таблица П6.1. *Назначения цифровых и аналоговых контактов популярных плат Arduino*

Arduino Uno с микроконтроллером ATmega168/328				**Arduino Mega с микроконтроллером ATmega1280/2560 (контакты 0–19)**		
Цифровой контакт	**Порт**	**Аналоговый контакт**	**Назначение**	**Порт**	**Аналоговый контакт**	**Назначение**
0	PD 0		USART RX	PE 0		USART0 RX, Контакт Int 8
1	PD 1		USART TX	PE 1		USART0 TX
2	PD 2		ExtInt 0	PE 4		**ШИМ** T3B, INT4
3	PD 3		**ШИМ** T2B, Ext Int 1	PE 5		**ШИМ** T3C, INT5
4	PD 4			PG 5		**ШИМ** T0B
5	PD 5		**ШИМ** T0B	PE 3		**ШИМ** T3A
6	PD 6		**ШИМ** T0A	PH 3		**ШИМ** T4A
7	PD 7			PH 4		**ШИМ** T4B
8	PB 0		Input capture	PH 5		**ШИМ** T4C
9	PB 1		**ШИМ** T1A	PH 6		**ШИМ** T2B
10	PB 2		**ШИМ** T1B, SS	PB 4		**ШИМ** T2A, Контакт Int 4

Таблица П6.1 (окончание)

Arduino Uno с микроконтроллером ATmega168/328				Arduino Mega с микроконтроллером ATmega1280/2560 (контакты 0–19)		
Цифровой контакт	Порт	Аналоговый контакт	Назначение	Порт	Аналоговый контакт	Назначение
11	PB 3		**ШИМ** T2A, MOSI	PB 5		**ШИМ** T1A, Контакт Int 5
12	PB 4		SPI MISO	PB 6		**ШИМ** T1B, Контакт Int 6
13	PB 5		SPI SCK	PB 7		**ШИМ** T0A, Контакт Int 7
14	PC 0	0		PJ 1		USART3 TX, Контакт Int 10
15	PC 1	1		PJ 0		USART3 RX, Контакт Int 9
16	PC 2	2		PH 1		USART2 TX
17	PC 3	3		PH 0		USART2 RX
18	PC 4	4	I^2C SDA	PD 3		USART1 TX, Ext Int 3
19	PC 5	5	I^2C SCL	PD 2		USART1 RX, Ext Int 2

***Таблица П6.2.** Назначение дополнительных контактов плат Mega*

Arduino Mega (контакты 20–44)			Arduino Mega (контакты 45–69)			
Цифровой контакт	Порт	Назначение	Цифровой контакт	Порт	Аналоговый контакт	Назначение
20	PD 1	I2C SDA, Ext Int 1	45	PL 4		**ШИМ** 5B
21	PD 0	I2C SCL, Ext Int 0	46	PL 3		**ШИМ** 5A
22	PA 0	Ext Memory addr bit 0	47	PL 2		T5 внешний счетчик
23	PA 1	Ext Memory bit 1	48	PL 1		ICP T5
24	PA 2	Ext Memory bit 2	49	PL 0		ICP T4
25	PA 3	Ext Memory bit 3	50	PB 3		SPI MISO
26	PA 4	Ext Memory bit 4	51	PB 2		SPI MOSI
27	PA 5	Ext Memory bit 5	52	PB 1		SPI SCK
28	PA 6	Ext Memory bit 6	53	PB 0		SPI SS
29	PA 7	Ext Memory bit 7	54	PF 0	0	
30	PC 7	Ext Memory bit 15	55	PF 1	1	
31	PC 6	Ext Memory bit 14	56	PF 2	2	

Таблица П6.2 *(окончание)*

Arduino Mega (контакты 20–44)			Arduino Mega (контакты 45–69)			
Цифровой контакт	**Порт**	**Назначение**	**Цифровой контакт**	**Порт**	**Аналоговый контакт**	**Назначение**
32	PC 5	Ext Memory bit 13	57	PF 3	3	
33	PC 4	Ext Memory bit 12	58	PF 4	4	
34	PC 3	Ext Memory bit 11	59	PF 5	5	
35	PC 2	Ext Memory bit 10	60	PF 6	6	
36	PC 1	Ext Memory bit 9	61	PF 7	7	
37	PC 0	Ext Memory bit 8	62	PK 0	8	Контакт Int 16
38	PD 7		63	PK 1	9	Контакт int 17
39	PG 2	ALE Ext Mem	64	PK 2	10	Контакт Int 18
40	PG 1	RD Ext Mem	65	PK 3	11	Контакт Int 19
41	PG 0	Wr Ext Mem	66	PK 4	12	Контакт Int 20
42	PL 7		67	PK 5	13	Контакт Int 21
43	PL 6		68	PK 6	14	Контакт Int 22
44	PL 5	**ШИМ** 5C	69	PK 7	15	Контакт Int 23

В табл. П6.3 приводится список режимов таймеров с соответствующими контактами, использующихся с популярными микроконтроллерами плат Arduino (работа с таймерами подробно рассматривается в *введении* в *главу 18* (*разд. 18.0*)).

Таблица П6.3. *Режимы таймеров*

Таймер	Arduino 168/328	Mega
Timer 0 mode (8-bit)	Быстрый ШИМ	Быстрый ШИМ
Timer0A контакт для `analogWrite()`	Контакт 6	Контакт 13
Timer0B контакт для `analogWrite()`	Контакт 5	Контакт 4
Timer 1 (16-разрядный)	ШИМ с фазовой коррекцией	ШИМ с фазовой коррекцией
Timer1A контакт для `analogWrite()`	Контакт 9	Контакт 11
Timer1B контакт для `analogWrite()`	Контакт 10	Контакт 12
Timer 2 (8-разрядный)	ШИМ с фазовой коррекцией	ШИМ с фазовой коррекцией
Timer2A контакт для `analogWrite()`	Контакт 11	Контакт 10
Timer2B контакт для `analogWrite()`	Контакт 3	Контакт 9
Timer 3 (16-разрядный)	Нет данных	ШИМ с фазовой коррекцией
Timer3A контакт для `analogWrite()`		Контакт 5
Timer3B контакт для `analogWrite()`		Контакт 2

***Таблица П6.3** (окончание)*

Таймер	Arduino 168/328	Mega
Timer3C контакт для `analogWrite()`		Контакт 3
Timer 4 (16-разрядный)	Нет данных	ШИМ с фазовой коррекцией
Timer4A контакт для `analogWrite()`		Контакт 6
Timer4B контакт для `analogWrite()`		Контакт 7
Timer4C контакт для `analogWrite()`		Контакт 8
Timer 5 (16-разрядный)	Нет данных	ШИМ с фазовой коррекцией
Timer5A контакт для `analogWrite()`		Контакт 46
Timer5B контакт для `analogWrite()`		Контакт 45
Timer5C контакт для `analogWrite()`		Контакт 44

Подробная информация по микроконтроллерам плат Arduino приведена в соответствующих справочных листках:

- для плат Arduino Uno и совместимых плат (микроконтроллер ATmega328): **https://oreil.ly/Y-12D**;
- для плат Mega и совместимых плат (микроконтроллер ATmega2560): **https://oreil.ly/uf2QD**.

Коды ASCII для стандартного и расширенного наборов символов

Сокращение ASCII означает American Standard Code for Information Interchange — Американский стандартный код для обмена информацией. Этот код является наиболее распространенным способом представления букв, цифр и других символов на компьютере. Каждый символ представляется кодом (числом). Например, число заглавной буквы *A* (английской) — 65, а прописной *a* — 97 (код прописных букв на 32 значения больше, чем код соответствующих заглавных букв).

Коды меньшие чем 32 называются *управляющими*. Изначально они были определены как невоспроизводимые при печати символы для управления ранними компьютерными терминалами вывода. В табл. П7.1 приводятся управляющие коды ASCII, которые наиболее часто используются в приложениях Arduino.

Таблица П7.1. *Часто употребляемые управляющие коды ASCII*

Десятичное значение	Шестнадцатеричное значение	Символ управляющей последовательности	Описание
0	0x0	'\0'	Символ Null (используется для завершения строки в языке C)
9	0x9	'\t'	Табуляция
10	0xA	'\n'	Новая строка
13	0xD	'\r'	Возврат каретки
27	0x1B		Клавиша <Escape>

В табл. П7.2 приводятся коды ASCII стандартного набора символов.

Таблица П7.2. *Таблица кодов ASCII стандартного набора символов*

Символ	Dec	Hex	Oct	Описание
	32	20	040	Пробел
!	33	21	041	Восклицательный знак
"	34	22	042	Кавычка (" в HTML)
#	35	23	043	Решетка (знак числа)
$	36	24	044	Доллар

Таблица П7.2 *(продолжение)*

Символ	Dec	Hex	Oct	Описание
%	37	25	045	Проценты
&	38	26	046	Амперсанд
'	39	27	047	Закрывающая одиночная кавычка (апостроф)
(	40	28	050	Открывающая скобка
)	41	29	051	Закрывающая скобка
*	42	2a	052	Звёздочка, умножение
+	43	2b	053	Плюс
,	44	2c	054	Запятая
-	45	2d	055	Дефис, минус
.	46	2e	056	Точка
/	47	2f	057	Наклонная черта (слеш, деление)
0	48	30	060	Ноль
1	49	31	061	Один
2	50	32	062	Два
3	51	33	063	Три
4	52	34	064	Четыре
5	53	35	065	Пять
6	54	36	066	Шесть
7	55	37	067	Семь
8	56	38	070	Восемь
9	57	39	071	Девять
:	58	3a	072	Двоеточие
;	59	3b	073	Точка с запятой
<	60	3c	074	Знак меньше
=	61	3d	075	Знак равенства
>	62	3e	076	Знак больше
?	63	3f	077	Знак вопроса
@	64	40	100	эт, собака
A	65	41	101	Заглавная A
B	66	42	102	Заглавная B
C	67	43	103	Заглавная C
D	68	44	104	Заглавная D
E	69	45	105	Заглавная E

Таблица П7.2 *(продолжение)*

Символ	Dec	Hex	Oct	Описание
F	70	46	106	Заглавная F
G	71	47	107	Заглавная G
H	72	48	110	Заглавная H
I	73	49	111	Заглавная I
J	74	4a	112	Заглавная J
K	75	4b	113	Заглавная K
L	76	4c	114	Заглавная L
M	77	4d	115	Заглавная M
N	78	4e	116	Заглавная N
O	79	4f	117	Заглавная O
P	80	50	120	Заглавная P
Q	81	51	121	Заглавная Q
R	82	52	122	Заглавная R
S	83	53	123	Заглавная S
T	84	54	124	Заглавная T
U	85	55	125	Заглавная U
V	86	56	126	Заглавная V
W	87	57	127	Заглавная W
X	88	58	130	Заглавная X
Y	89	59	131	Заглавная Y
Z	90	5a	132	Заглавная Z
[	91	5b	133	Открывающая квадратная скобка
\	92	5c	134	Обратная наклонная черта (обратный слеш)
]	93	5d	135	Закрывающая квадратная скобка
^	94	5e	136	Циркумфлекс, возведение в степень, знак вставки
_	95	5f	137	Нижнее подчёркивание
`	96	60	140	Открывающая одиночная кавычка, гравис, знак ударения
a	97	61	141	Строчная a
b	98	62	142	Строчная b
c	99	63	143	Строчная c
d	100	64	144	Строчная d
e	101	65	145	Строчная e

Таблица П7.2 *(окончание)*

Символ	Dec	Hex	Oct	Описание
f	102	66	146	Строчная f
g	103	67	147	Строчная g
h	104	68	150	Строчная h
i	105	69	151	Строчная i
j	106	6a	152	Строчная j
k	107	6b	153	Строчная k
l	108	6c	154	Строчная l
m	109	6d	155	Строчная m
n	110	6e	156	Строчная n
o	111	6f	157	Строчная o
p	112	70	160	Строчная p
q	113	71	161	Строчная q
r	114	72	162	Строчная r
s	115	73	163	Строчная s
t	116	74	164	Строчная t
u	117	75	165	Строчная u
v	118	76	166	Строчная v
w	119	77	167	Строчная w
x	120	78	170	Строчная x
y	121	79	171	Строчная y
z	122	7a	172	Строчная z
{	123	7b	173	Открывающая фигурная скобка
|	124	7c	174	Вертикальная черта
}	125	7d	175	Закрывающая фигурная скобка
~	126	7e	176	Тильда (приблизительно)

Коды ASCII со значениями выше 128 (ANSI) используются для представления расширенного набора символов, включающего буквы иных алфавитов, чем английский, и специальные символы. Эти коды приводятся в табл. П7.3.

Таблица П7.3. *Расширенный набор символов (ANSI) в русской кодировке Win-1251*

Символ	Dec	Hex	Oct	Описание
Ђ	128	80	200	
Ѓ	129	81	201	

***Таблица П7.3** (продолжение)*

Символ	Dec	Hex	Oct	Описание
‚	130	82	202	Запятая
ѓ	131	83	203	
„	132	84	204	Кавычка нижняя правая
…	133	85	205	Многоточие
†	134	86	206	Одинарный крест
‡	135	87	207	Двойной крест
€	136	88	210	Знак евро
‰	137	89	211	Промилле (тысячная часть числа)
Љ	138	8A	212	
‹	139	8B	213	Знак меньше
Њ	140	8C	214	
Ќ	141	8D	215	
Ћ	142	8E	216	
Џ	143	8F	217	
ђ	144	90	220	
‘	145	91	221	Левая одинарная верхняя кавычка
’	146	92	222	Правая верхняя одинарная кавычка
“	147	93	223	Левые двойные верхние кавычки
”	148	94	224	Правые двойные верхние кавычки
•	149	95	225	Крупная точка
–	150	96	226	Короткий дефис (тире)
—	151	97	227	Длинный дефис (тире)
	152	98	230	
™	153	99	231	Торговый знак
љ	154	9A	232	
›	155	9B	233	Знак больше
њ	156	9C	234	
ќ	157	9D	235	
ћ	158	9E	236	
џ	159	9F	237	
	160	A0	240	
Ў	161	A1	241	

***Таблица П7.3** (продолжение)*

Символ	Dec	Hex	Oct	Описание
ў	162	A2	242	
J	163	A3	243	
¤	164	A4	244	Валюта
Ґ	165	A5	245	
¦	166	A6	246	
§	167	A7	247	Параграф, раздел
Ё	168	A8	250	Заглавная Ё
©	169	A9	251	Копирайт
Є	170	AA	252	
«	171	AB	253	Левые двойные кавычки
¬	172	AC	254	
	173	AD	255	Дефис
®	174	AE	256	Зарегистрированный торговый знак
Ї	175	AF	257	Английская прописная (большая) I с умлаутом (двумя точками наверху)
°	176	B0	260	Градус
±	177	B1	261	Плюс-минус
I	178	B2	262	
і	179	B3	263	
ґ	180	B4	264	
µ	181	B5	265	Мю
¶	182	B6	266	Enter, знак абзаца
·	183	B7	267	Средняя квадратная точка
ё	184	B8	270	Строчная ё
№	185	B9	271	Порядковый номер
є	186	BA	272	
»	187	BB	273	Правые двойные кавычки
ј	188	BC	274	
S	189	BD	275	
s	190	BE	276	
ї	191	BF	277	Английская строчная (маленькая) буква i, но только не с одной, а с двумя точками сверху (умлаут)
А	192	C0	300	Заглавные буквы русского алфавита

Таблица П7.3 *(продолжение)*

Символ	Dec	Hex	Oct	Описание
Б	193	C1	301	
В	194	C2	302	
Г	195	C3	303	
Д	196	C4	304	
Е	197	C5	305	
Ж	198	C6	306	
З	199	C7	307	
И	200	C8	310	
Й	201	C9	311	
К	202	CA	312	
Л	203	CB	313	
М	204	CC	314	
Н	205	CD	315	
О	206	CE	316	
П	207	CF	317	
Р	208	D0	320	
С	209	D1	321	
Т	210	D2	322	
У	211	D3	323	
Ф	212	D4	324	
Х	213	D5	325	
Ц	214	D6	326	
Ч	215	D7	327	
Ш	216	D8	330	
Щ	217	D9	331	
Ъ	218	DA	332	
Ы	219	DB	333	
Ь	220	DC	334	
Э	221	DD	335	
Ю	222	DE	336	
Я	223	DF	337	
а	224	E0	340	Строчные буквы русского алфавита

Таблица П7.3 *(окончание)*

Символ	Dec	Hex	Oct	Описание
б	225	E1	341	
в	226	E2	342	
г	227	E3	343	
д	228	E4	344	
е	229	E5	345	
ж	230	E6	346	
з	231	E7	347	
и	232	E8	350	
й	233	E9	351	
к	234	EA	352	
л	235	EB	353	
м	236	EC	354	
н	237	ED	355	
о	238	EE	356	
п	239	EF	357	
р	240	F0	360	
с	241	F1	361	
т	242	F2	362	
у	243	F3	363	
ф	244	F4	364	
х	245	F5	365	
ц	246	F6	366	
ч	247	F7	367	
ш	248	F8	370	
щ	249	F9	371	
ъ	250	FA	372	
ы	251	FB	373	
ь	252	FC	374	
э	253	FD	375	
ю	254	FE	376	
я	255	FF	377	

Коды ASCII обоих наборов символов можно просмотреть в окне монитора порта с помощью скетча из листинга П7.1.

Листинг П7.1. Скетч для вывода в окно монитора порта кодов ASCII всех наборов символов

```
/*
* Скетч ASCII
*Отображает на экране символы с кодами ASCII on 1 до 255
*/

void setup()
{
   Serial.begin(9600);
   while(!Serial); // Для плат Leonardo и других 32-разрядных плат
   for(int i=1; i < 256; i++)
   {
      Serial.write(i);
      Serial.print(", dec: ");
      Serial.print(i, DEC);
      Serial.print(", hex: ");
      Serial.println(i, HEX);
   }
}

void loop()
{
}
```

Обратите внимание, что некоторые устройства, например жидкокристаллические дисплеи (см. *главу 11*), могут использовать другие символы для кодов выше 128. Поэтому обратитесь к справочному листку на свое устройство, чтобы узнать, какие именно символы оно поддерживает.

Предметный указатель

A

X

Z

A

Б

В

Г

Д

Ж

З

И

К

Л

М

Н

О

П

P

C

Т

У

Ф

Ц

Ч

Ш

Э

Я

Об авторах

Майкл Марголис (Michael Margolis) — специалист в области вычислений реального времени и эксперт по разработке и внедрению аппаратных и программных решений для взаимодействия с окружающей средой. Он свыше 30 лет работает на должностях высшего уровня в компаниях Sony, Microsoft и Lucent/Bell Labs. Является автором библиотек и программного ядра, входящих в состав официального дистрибутива Arduino 1.0.

Брайан Джепсон (Brain Jepson) — работает контент-менеджером в компании LinkedIn Learning, где руководит курсами инженерно-технического проектирования. Он также является одним из организаторов сообщества Providence Geeks, одним из основателей команды планирования и постановки ярмарки/выставки National Maker Faire, а также сопродюсером ярмарки/выставки Providence Mini Maker Faire. Брайан распространяет знания в области сборки электронных и цифровых устройств путем организации практических мероприятий и семинаров, тесно сотрудничая с общественным некоммерческим художественным центром AS220 и с компьютерным музеем штата Род-Айленд (обе эти некоммерческие организации базируются в штате Род-Айленд).

Ник Уэлдин (Nick Weldin) — работает в центре Rix Centre Университета Восточного Лондона, проводя исследования в области технологий, которые могут помочь людям, испытывающим трудности с обучением, принимать участие в онлайновых событиях и работать с компьютером. Он также занимает должность старшего специалиста в компании Tinker It!, работая над различными технологическими проектами, часто связанными с платформой для создания прототипов Arduino на основе гибкого и простого в использовании открытого программного и аппаратного обеспечения. В число других его проектов входит работа с организацией Paddington Arts, театральной компанией Oily Cart Theatre Company и театром Deafinitely Theatre.

Об обложке

На обложке третьего издания этой книги[1] изображен игрушечный кролик. Механические игрушки, подобные этой, приводятся в действие с помощью пружин, шестеренок, шкивов, рычажков и других простых механизмов с механическим источником энергии. У этих игрушек долгая история — до наших времен дошли образцы, сделанные в Древней Греции, Китае и арабских странах.

Производство механических игрушек процветало на ранних этапах современного периода европейской истории. Еше в начале XV века немецкий изобретатель Карел Грод (Karel Grod) демонстрировал летающие заводные игрушки. Выдающиеся ученые того времени, такие как Леонардо да Винчи, Рен Декарт и Галилео Галилей, отметились в том числе и своим увлечением созданием механических игрушек. Знаменитый механический лев Леонардо да Винчи, сделанный им в 1509 году для короля Людовика XII, сам подошел к королю и, открыв створки на своей груди, представил на всеобщее обозрение лилию — эмблему королевского дома Франции.

Считается, что искусство создания механических игрушек достигло своей вершины к концу XVIII века, когда появились знаменитые автоматоны, изготовленные швейцарским часовщиком Пьером Жаке-Дрозом (Pierre Jaquet-Droz) и его сыном Генри-Луи (Henri-Louis). Их механические изделия, созданные в виде человеческих фигур, выполняли, как живые, различные присущие человеку действия: макали перо в чернильницу, писали полные предложения, чертили рисунки и сдували со страницы крошки стирательной резинки. В XIX столетии европейские и американские компании также выпускали популярные заводные игрушки, за которыми в наше время охотятся коллекционеры.

Поскольку эти игрушки представляли собой сложные механизмы и были изысканно украшены, изготовление их занимало долгое время, и стоили они очень дорого, вследствие чего развлекаться ими могли позволить себе только королевские особы или обеспеченные взрослые. Только ближе к концу XIX века, с появлением массового производства и дешевых материалов (жести и, позже, пластмасс), механические игрушки стали доступны и для детей. Эти недорогие двигающиеся устройства были популярными около столетия, пока не были вытеснены игрушками на батарейках.

Иллюстрация на обложке выполнена Карен Монтгомери (Karen Montgomery) на основе черно-белой гравюры из Доверского архива иллюстраций (Dover Pictorial Archive).

---

[1] Имеется в виду английское исходное издание.

O'REILLY®

Arduino

БОЛЬШАЯ **книга рецептов**

Хотите создавать устройства, способные взаимодействовать с внешним миром? Этот справочник будет идеальным пособием для любого, кто хочет экспериментировать с популярной платформой Arduino. Он содержит подробное описание решений свыше 200 практических задач по созданию различных устройств и приспособлений, включая проекты для Интернета вещей, мониторинга окружающей среды, системы для определения местонахождения и положения в пространстве, а также устройств, которые реагируют на касание, звук, тепло и освещение.

Все примеры третьего издания обновлены для версии 1.8 среды Arduino IDE. Каждое решение включает в себя программный код с подробными комментариями, его анализ и обсуждение возможных проблем. Такой подход позволит вам сразу начать создавать, расширять и улучшать свои проекты, независимо от уровня технической подготовки, будь вы инженер, разработчик, деятель искусств, студент или «мейкер-самоделкин».

«Эта книга будет прекрасным ресурсом для тех, кто приступает к разработке проектов на основе платформы Arduino. Содержащиеся в ней примеры решений доступны для понимания начинающих, но также полезны и опытным разработчикам».

Дон Коулман (Don Coleman),
главный директор по инновациям,
компания Chariot Solutions

Книга поможет вам

- Быстро войти в курс дела по работе с платой Arduino и научиться применять основные приемы программирования
- Изучить основные способы считывания цифровых и аналоговых сигналов
- Использовать платформу Arduino с различными популярными датчиками и другими устройствами ввода
- Выводить текст и графику на дисплеи, генерировать звуки и управлять электродвигателями разных типов
- Дистанционно управлять устройствами, включая телевизоры и бытовые приборы
- Обучиться методам измерения времени и использования пауз в исполнении кода
- Применять продвинутые методы программирования и управления памятью

Майкл Марголис (Michael Margolis) работает в области вычислений реального времени и имеет свыше 30 лет опыта на руководящих должностях в компаниях Sony, Microsoft и Lucent/Bell Labs.

Брайан Джепсон (Brain Jepson) работает контент-менеджером в компании LinkedIn Learning, где руководит курсами инженерно-технического проектирования.

Николас Роберт Уэлдин (Nicholas Robert Weldin) работает в британском благотворительном центре исследований и разработок Rix Centre. Проводит исследования в области мультимедийных технологий, которые помогают людям с ограниченными возможностями.

ISBN 978-5-9775-6687-2

9 785977 566872

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

интернет-магазин: bhv.ru
facebook: bhvPetersburg
вконтакте: bhvspb

bhv®