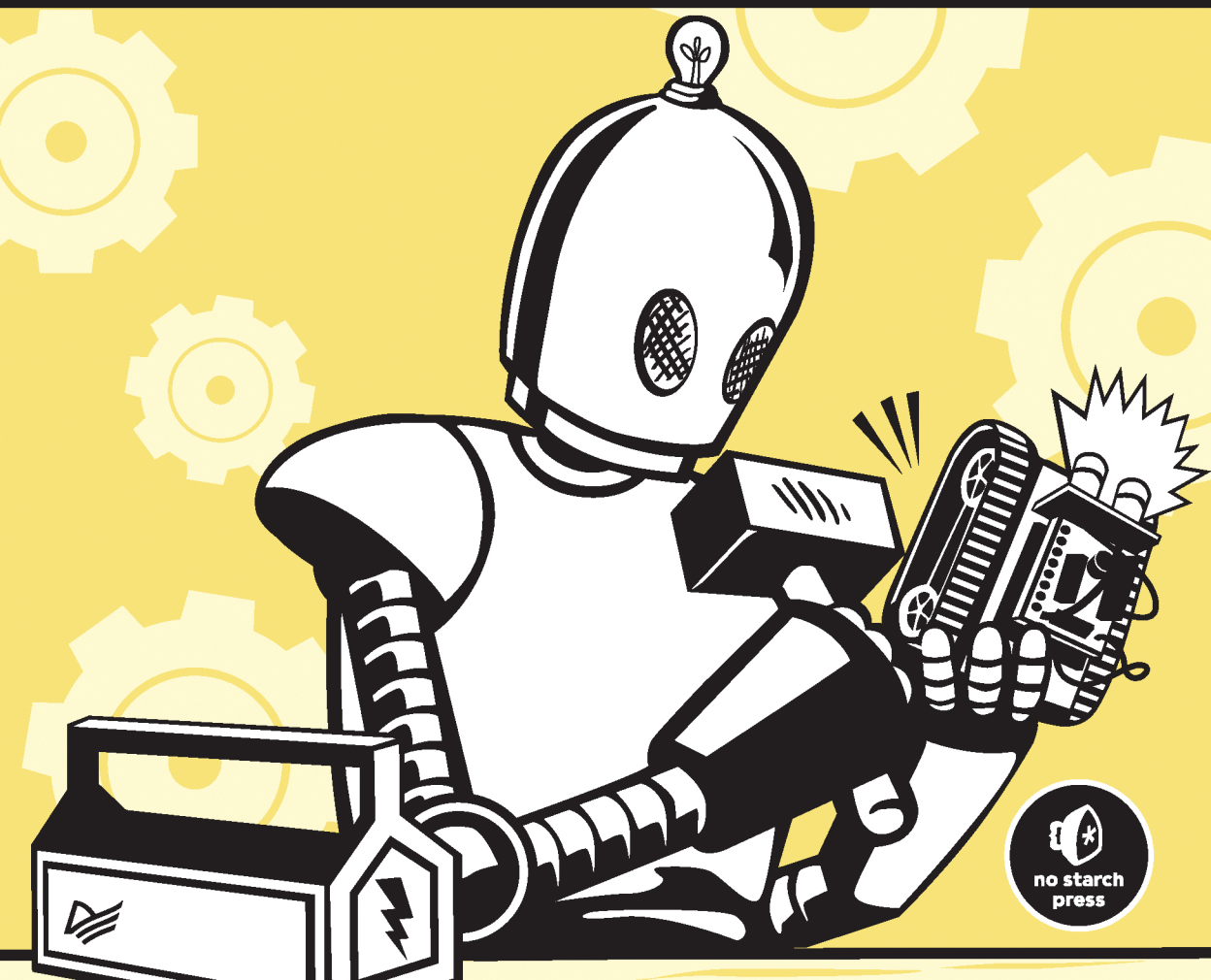
ВТОРОЕ ИЗДАНИЕ

ИЗУЧАЕМ ARDUINO

65 ПРОЕКТОВ СВОИМИ РУКАМИ

ДЖОН БОКСЕЛЛ



no starch press

ARDUINO WORKSHOP

2nd Edition

A Hands-on Introduction with 65 Projects

John Boxall

no starch press

San Francisco

ИЗУЧАЕМ ARDUINO

ВТОРОЕ ИЗДАНИЕ

65 ПРОЕКТОВ СВОИМИ РУКАМИ

ДЖОН БОКСЕЛЛ

ПИТЕР®

Санкт-Петербург • Москва • Минск

2022

ББК 32.973.23-018.2
УДК 004.9
Б78

Бокселл Джон

Б78 Изучаем Arduino. 65 проектов своими руками. 2-е изд. — СПб.: Питер, 2022. — 448 с.: ил.

ISBN 978-5-4461-1918-9

Что такое Arduino? За этим словом прячется легкое и простое устройство, которое способно превратить кучу проводов и плат в робота, управлять умным домом и многое другое. Разнообразие устройств ввода/вывода — датчиков, индикаторов, дисплеев и электромоторов — позволяет создавать самые невероятные проекты.

Второе издание этой книги было полностью переработано, ведь технологии не стоят на месте. Познакомившись с основами Arduino, вы сможете экспериментировать с сенсорными экранами и жидкокристаллическими дисплеями, займетесь робототехникой, освоите работу с датчиками и беспроводной передачей данных и научитесь дистанционно управлять устройствами с помощью телефона.

В мире продано уже более 35 000 экземпляров этой книги.

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

ББК 32.973.23-018.2
УДК 004.9

ISBN 978-1718500587 англ.

ISBN 978-5-4461-1918-9

Краткое содержание

Оглавление

Маме и моей дорогой Кэтлин,
всегда веривших в меня.

Об авторе

Джон Бокселл (John Boxall) более 26 лет занимается разработкой, распространением и продажей электроники. Последние годы в свободное время он пишет учебные пособия, проекты и делает обзоры об Arduino.

О научном редакторе

Ксандер Солдаат (Xander Soldaat) — бывший партнер по связям с сообществом Mindstorms в LEGO MINDSTORMS. Прежде чем стать штатным разработчиком программного обеспечения в Robomatter, а потом в VEX Robotics, 18 лет был архитектором и инженером ИТ-инфраструктуры. Сейчас трудится инженером-исследователем в сфере разработки встроенных решений Wi-Fi. В свободное время Ксандер любит возиться с роботами и домашними ретрокомпьютерами, увлекается 3D-печатью.

От издательства

Благодарим Владимира Трондина за помощь в подготовке издания.

Владимир окончил Иркутский университет по специальности «Радиофизика и электроника» в 1984 году. В 2002–2004 годах работал инженером-разработчиком программного обеспечения для встроенных процессоров (Embedded Software Engineer) в Polar Electro OY. С 2004 года инженер по применению (Applications Engineer) в National Semiconductor Finland OY и Texas Instrument Finland OY после приобретения National Texas в 2011 году.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Благодарности

Прежде всего я хотел бы выразить огромную благодарность коллективу разработчиков Arduino: Массимо Банци (Massimo Banzi), Дэвиду Куартиллье (David Cuartielles), Тому Иго (Tom Igoe), Джанлуке Мартино (Gianluca Martino) и Дэвиду Меллису (David Mellis). Без вашей прозорливости, идей и упорного труда не было бы этого проекта.

Благодарю всех читателей первого издания за их отзывы и конструктивную критику.

Огромное спасибо моему научному редактору Ксандеру Солдаату (Xander Soldaat) за его вклад в этот большой проект и настойчивость в доведении книги до логического конца.

Хочу выразить глубокую признательность за понимание и поддержку следующим компаниям: Adafruit, Keysight Technologies, Freetronics, PMD Way, Seeed Studio, Sharp Corporation, SparkFun и Tronixlabs.

Кроме того, мне хочется отдельно поблагодарить Freetronics PMD Way за предоставление великолепных электронных компонентов. Спасибо всем тем, кто потратил свое время на создание библиотек для Arduino, которые упрощают работу многим разработчикам.

Выражаю свое уважение и благодарность команде Fritzing за их превосходный открытый инструмент проектирования электронных схем, которым я пользовался во время работы над книгой.

Спасибо всем, кто вдохновлял и поддерживал меня: Элизабет Прайс (Elizabeth Pryce), Джонатану Оксеру (Jonathan Oxer), Филипу Линдси (Philip Lindsay), Кену Ширрифу (Ken Shirriff), Натану Кеннеди (Nathan Kennedy) и Дэвиду Джонсу (David Jones).

И наконец, хочу сказать спасибо всем сотрудникам издательства No Starch Press за их труд над этим обновленным изданием: Патрику Диджусто (Patrick DiJusto) за редакторский вклад, Дапиндеру Досанжу (Dapinder Dosanjh) и Натану Хайдельбергеру (Nathan Heidelberger) за их бесконечное терпение, Рэйчел Монаган (Rachel Monaghan) за управление всем процессом издания книги, Пауле Флеминг (Paula Fleming) за литературную правку, Рэйчел Хед (Rachel Head) за корректуру, Джоанне Бурек (JoAnne Burek) за создание предметного указателя. Отдельная благодарность Биллу Поллоку (Bill Pollock) за поддержку, советы и способность убедить, что иногда авторское объяснение темы — это не лучшее и не единственно верное решение.

1

Введение

Приходилось ли вам, разглядывая какое-нибудь устройство, задумываться над тем, как оно работает *в действительности*? Возможно, это был катер на дистанционном управлении, лифт, автомат с напитками или электронная игрушка? А может быть, вам хотелось самому создать робота или придумать электронное управление для модели железной дороги? Или, возможно, вы бы хотели в течение длительного времени собирать данные о погоде и анализировать их? Как и с чего вы могли бы начать свой проект?

Плата Arduino (рис. 1.1) поможет на практике раскрыть некоторые секреты электроники. Arduino, созданная Массимо Банци и Дэвидом Куартиллье, предлагает бюджетный способ создания интерактивных проектов и таких объектов, как дистанционно управляемые роботы, GPS-трекеры и электронные игры.

Рис. 1.1. Плата Arduino

Проект Arduino рос экспоненциально с момента рождения в 2005 году. Теперь это процветающая индустрия, поддерживаемая множеством людей, объединенных общим стремлением к поиску нового. Здесь вы найдете и отдельных энтузиастов, и целые группы, от небольших клубов и местных кружков по интересам до объединений специалистов и образовательных учреждений, заинтересованных в использовании этой платформы.

Узнать о разнообразии проектов на основе Arduino вы можете, просто воспользовавшись поисковиком. Уверен, вы найдете невероятное количество проектов, блогов и статей.

Возможности безграничны

Быстро пролистав эту книгу, вы увидите, что плату Arduino можно использовать для создания устройств, от самых простых, незатейливо мигающих светодиодами, до очень сложных, взаимодействующих со смартфоном, и множества проектов, по сложности находящихся между ними.

Взгляните на Wi-Fi-метеостанцию, сконструированную Бекки Стерн (Becky Stern). На рис. 1.2 вы можете увидеть разные примеры вывода информации на ней. Это устройство работает на Arduino-совместимой плате с интерфейсом Wi-Fi. Оно показывает цифры максимальной дневной температуры и отображает прогноз погоды на день с помощью треугольников, меняющих цвета.

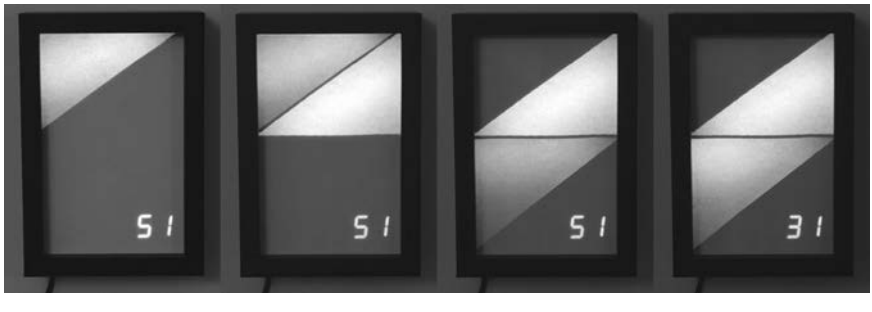


Рис. 1.2. Разные примеры отображения прогноза погоды на табло устройства[1]

Благодаря простоте получения различной информации в интернете вы можете с помощью этого устройства отображать не только прогноз погоды, но и другие данные. Подробнее об этом вы можете узнать по ссылке https://www.instructables.com/id/WiFi-Weather-Display-With-ESP8266/.

[1] Температура в градусах Фаренгейта. — *Примеч. пер.*

Не хотите ли сделать собственный классический компьютер из прошлого? Благодаря возможностям процессора Arduino можно эмулировать подобные устройства. На рис. 1.3 показан один из примеров — устройство KIM Uno Оскара Вермёлена (Oscar Vermeulen), имитирующее компьютер KIM-1 1976 года. Больше информации вы найдете на странице https://en.wikipedia.org/wiki/KIM-1.

Рис. 1.3. Эмулятор компьютера KIM-1 на Arduino

Поработав над этим проектом, можно понять принцип функционирования первых процессоров и получить базовые знания, необходимые для понимания современных компьютеров. Вы сможете воспроизвести Kim Uno, потратив меньше 50 долларов. Благодаря такой низкой цене подобными проектами можно поделиться с другими энтузиастами. Дополнительную информацию вы найдете на странице https://obsolescence.wixsite.com/obsolescence/kim-uno-summary-c1uuh/.

А вот еще один пример: Михалис Василакис (Michalis Vasilakis) обожает вручную создавать недорогие инструменты. Отличный пример — его мини-плоттер с ЧПУ на основе Arduino (рис. 1.4). В этом проекте используются плата Arduino, механизмы от старых приводов компакт-дисков и другие недорогие детали для создания устройства с числовым программным управлением (ЧПУ), которое может рисовать

с высокой точностью на плоской поверхности. Больше информации вы найдете на странице http://www.ardumotive.com/new-cnc-plotter.html.

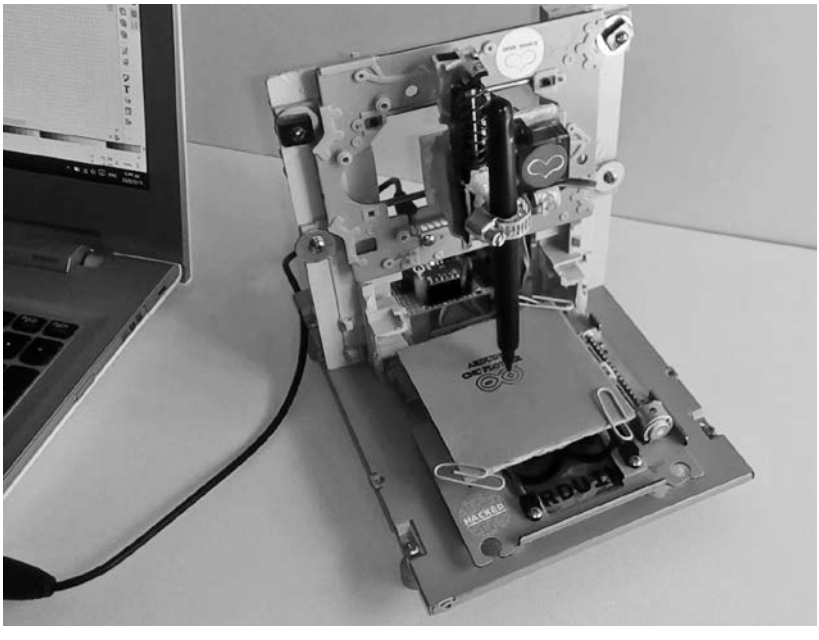


Рис. 1.4. Мини-плоттер с ЧПУ на основе Arduino

Это лишь несколько случайных примеров, показывающих возможности применения Arduino. Уже сейчас вы сможете создавать собственные простые проекты, а после прочтения этой книги — и более сложные.

Сила в массовости

Если вы сторонник социального обучения, то найдите в сети местных единомышленников и форумы энтузиастов Arduino, чтобы увидеть, что они делают, используя Arduino. Члены этих групп помогут взглянуть на мир Arduino с творческой стороны. Во многих сообществах создаются маленькие Arduino-совместимые платы. Там вы найдете массу интересного, познакомитесь с новыми людьми и поделитесь своими знаниями об Arduino с другими участниками.

ПРИМЕЧАНИЕ

На веб-странице https://nostarch.com/arduino-workshop-2nd-edition/ вы сможете загрузить файлы скетчей и найти дополнения и обновления к тексту книги.

Компоненты и аксессуары

Как и любые электронные устройства, плату Arduino можно приобрести в розницу, а продавцы предложат вам широкий выбор компонентов и аксессуаров. Я рекомендую во время покупок выбирать оригинальные версии Arduino или качественные копии, иначе вы рискуете купить неисправный товар. Не рискуйте, приобретая плату неизвестного производителя. Это увеличит как финансовые, так и физические и временные затраты на реализацию проекта. Список авторизованных распространителей Arduino можно найти по адресу http://arduino.cc/en/Main/Buy/.

Ниже приводится список поставщиков (в алфавитном порядке), к которым я рекомендую обращаться, если нужно купить компоненты и аксессуары для создания проектов на основе Arduino:

- Adafruit Industries (http://www.adafruit.com/);
- Arduino Store USA (https://store.arduino.cc/usa/);
- PMD Way (https://pmdway.com/);
- SparkFun Electronics (http://www.sparkfun.com/).

Список компонентов из этой книги, обновления и дополнения к тексту можно найти на сайте https://nostarch.com/arduino-workshop-2nd-edition/. Но пока отложите поход в магазин и ознакомьтесь с первыми главами, чтобы понять, что же потребуется в первую очередь, и не тратить деньги на ненужное.

Необходимое программное обеспечение

Итак, программировать платы Arduino можно почти на любом компьютере, но важно иметь программное обеспечение (ПО), которое называют *интегрированной средой разработки* (Integrated Development Environment, IDE). Чтобы запустить это программное обеспечение, нужен компьютер, подключенный к интернету, с установленной на нем одной из следующих операционных систем:

- Mac OS X или выше;
- Windows 10 Home 32- или 64-разрядной или выше;
- Linux 32- или 64-разрядной (Ubuntu или подобной ей).

Разработка IDE ведется с 2005 года, и к настоящему времени широко используются версии 2.*x* (точное число *x* в номере версии у всех может быть разным, но сами инструкции из этой книги остаются актуальными). По сравнению с версией 1.*x*, у версии 2.*x* есть некоторые функции, упрощающие разработку скетчей, включая интерактивное автодополнение, улучшенную навигацию и более удобные средства управления платами и библиотеками. Кроме того, для некоторых плат отладчик позволяет запускать и останавливать выполнение скетча в интерактивном режиме.

Но если вы новичок в мире Arduino, вам об этом беспокоиться не нужно. Просто помните, что команда Arduino всегда работает над улучшением инструментария.

А сейчас самое время загрузить и установить IDE. Для этого перейдите к разделу с названием вашей операционной системы и следуйте инструкциям по установке. Не забудьте вместе с платой Arduino купить подходящий кабель USB. Даже если у вас еще нет этой платы, все равно загрузите IDE и займитесь ее исследованием.

macOS

В этом разделе вы найдете инструкции по загрузке и настройке Arduino IDE в macOS.

1. Откройте страницу загрузки (https://www.arduino.cc/en/software/) и загрузите последнюю доступную версию IDE для вашей операционной системы.
2. Дважды щелкните на файле `.dmg` в папке Downloads (Загрузки). Когда откроется окно установки, перетащите значок Arduino в папку Applications (Приложения).
3. Откройте IDE (рис. 1.5).

Рис. 1.5. Окно IDE в macOS

4. Теперь нужно настроить IDE для использования платы Arduino Uno. Щелкните на верхнем значке на левой боковой панели окна IDE, чтобы открыть Boards Manager (Менеджер плат). Найдите вариант с названием платы Arduino Uno и нажмите кнопку Install (Установить).

5. Разверните список в верхней части окна IDE, где виден текст no board selected (плата не выбрана), и выберите пункт Select Other Board & Port (Выбрать другую плату и порт). Затем выберите Arduino Uno из предложенного списка плат.

   Вам может быть предложено установить пакет инструментов командной строки для разработчика (Command Line Developer tools) — установите его.

Теперь аппаратное и программное обеспечение готовы к работе, и можно перейти к разделу «Безопасность».

Windows 10

Здесь вы найдете инструкции по загрузке и настройке Arduino IDE в Windows.

1. Откройте страницу загрузки (http://arduino.cc/en/software/) и загрузите последнюю доступную версию IDE для вашей операционной системы.
2. Браузер может предложить выбор между сохранением и запуском загруженного файла. Выберите Run (Запустить), чтобы установка началась автоматически по завершении загрузки. Если этого не произошло, после окончания загрузки отыщите файл `Arduino.exe` в папке Downloads (Загрузки) и запустите его для установки IDE. После завершения установки запустите IDE.
3. Теперь нужно настроить IDE на использование платы Arduino Uno. Щелкните на верхнем значке на боковой панели слева в окне IDE, чтобы открыть Boards Manager (Менеджер плат). Отыщите вариант с названием платы Arduino Uno и нажмите кнопку Install (Установить).
4. Разверните список в верхней части окна IDE, где виден текст no board selected (плата не выбрана), и выберите пункт Select Other Board & Port (Выбрать другую плату и порт). Затем выберите Arduino Uno из предложенного списка плат.

Теперь аппаратное и программное обеспечение готовы к работе, и можно перейти к разделу «Безопасность».

Ubuntu Linux

Если вы пользуетесь Ubuntu Linux, ниже вы найдете инструкции по загрузке и настройке Arduino IDE в этой среде.

Для установки IDE следуйте указаниям ниже.

1. Откройте страницу загрузки (https://www.arduino.cc/en/software/) и загрузите последнюю доступную версию IDE для вашей операционной системы.
2. В диалоге выберите вариант Save File (Сохранить файл) и нажмите OK.

3. По завершении загрузки найдите файл `.zip` в папке Downloads (Загрузки), откройте его в Archive Manager (Диспетчер архивов) и извлеките содержимое на рабочий стол.
4. Перейдите в папку с извлеченным программным обеспечением и в окне терминала введите команду `./arduino-ide` для запуска IDE.
5. Теперь нужно настроить IDE. Для этого подключите плату Arduino к компьютеру с помощью кабеля USB.
6. Выберите пункт меню Tools ▸ Port (Инструменты ▸ Порт) и затем порт /dev/ttyACM*x* в подменю, где *x* — цифра (для выбора должен быть доступен только один порт с подобным именем[1]).

Теперь аппаратное и программное обеспечение готовы к работе.

Безопасность

Чем бы вы ни занимались, нужно сначала обезопасить себя и окружающих. Как вы потом увидите, я буду использовать в работе простые ручные инструменты, электрические устройства с питанием от аккумуляторов, разные ножи и резаки и иногда — паяльник. Ни в одном из представленных далее проектов вам не придется работать с напряжением обычной электросети. Оставим это специалистам-электрикам. Но не забывайте, что нельзя прикасаться к оголенным проводам под высоким напряжением!

Что дальше?

Вы собираетесь пуститься в увлекательное путешествие, где будете создавать устройства, прежде казавшиеся фантастическими. В этой книге вас ждут 65 проектов на основе Arduino, от очень простых до относительно сложных. Все они были спроектированы специально, чтобы помочь вам научиться создавать что-нибудь полезное. Итак, в путь!

[1] Для некоторых клонов порт будет выглядеть как dev/ttyUSBx. — *Примеч. науч. ред.*

2

Знакомство с платой Arduino и IDE

В этой главе мы займемся исследованием платы Arduino и среды разработки Arduino IDE, которая будет использоваться для создания *скетчей* (так в мире Arduino называются программы) и загрузки их в плату Arduino. Вы познакомитесь с базовой структурой скетча и основными функциями, а также создадите и загрузите свой первый скетч.

Плата Arduino

Что такое Arduino? Согласно определению на сайте Arduino (http://www.arduino.cc/), это *открытая аппаратная платформа для макетирования электронных устройств, основанная на гибком и простом в использовании аппаратном и программном обеспечении. Она предназначена для всех, кто интересуется созданием интерактивных устройств*.

Проще говоря, Arduino — это маленький компьютер, который можно запрограммировать для взаимодействия с разными физическими объектами с помощью разных входных и выходных сигналов. Основная модель Arduino, **Uno**, небольшая и может легко поместиться на ладони, как можно видеть на рис. 2.1.

На первый взгляд плата выглядит не очень внушительно, но она позволяет создавать устройства, взаимодействующие с окружающим миром. Используя практически неограниченный спектр устройств ввода и вывода, таких как датчики, индикаторы, дисплеи, электромоторы и т. д., вы сможете запрограммировать любые взаимодействия для создания функционального устройства. Например, художник может создать светодиодную инсталляцию, мигающую в такт движениям проходящих мимо посетителей, студенты — сконструировать автономного робота, который

будет обнаруживать открытый огонь и гасить его, а синоптики — спроектировать систему измерения температуры и влажности и передавать эти данные в свои системы в виде текстовых сообщений. Еще больше примеров вы сможете найти, просто поискав в интернете.

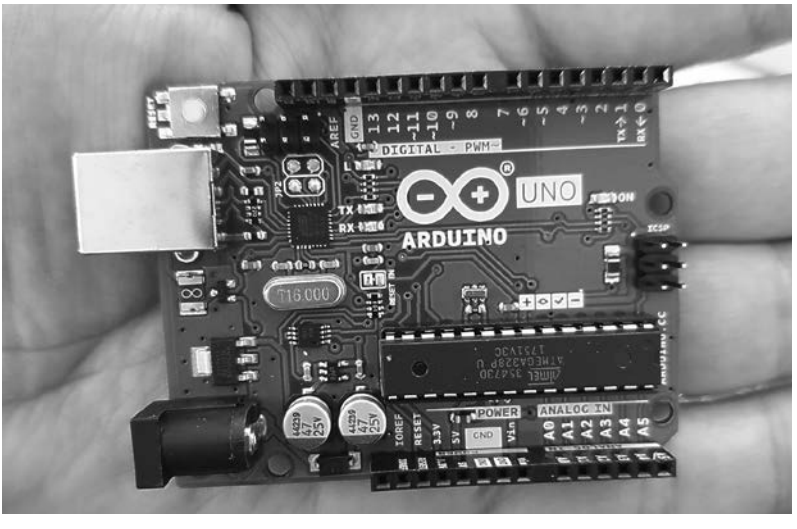


Рис. 2.1. Небольшая плата Arduino Uno

Теперь подробнее исследуем *аппаратную* часть Arduino Uno (ее физическое устройство). Если вы не поймете что-то в этой главе, не расстраивайтесь, потому что все, о чем будет рассказываться, мы детально разберем в следующих.

Осмотрите плату Uno со всех сторон. Поверните ее к себе стороной с разъемами, как показано на рис. 2.2.

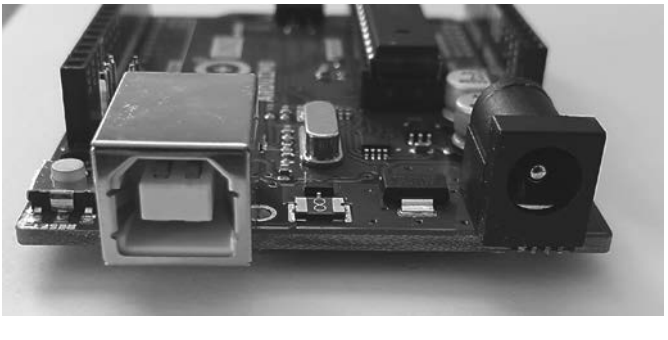


Рис. 2.2. Разъем USB и разъем подключения питания

Слева находится разъем подключения универсальной последовательной шины (Universal Serial Bus, USB). С его помощью плата подключается к компьютеру, он выполняет такие три функции, как питание устройства, загрузка ваших скетчей с инструкциями и обмен данными с компьютером. Справа находится разъем подключения блока питания. С его помощью можно подключить плату к стандартному блоку питания (понижающему напряжение до 5 В)[1].

В центре, чуть ниже середины, находится микроконтроллер (рис. 2.3).

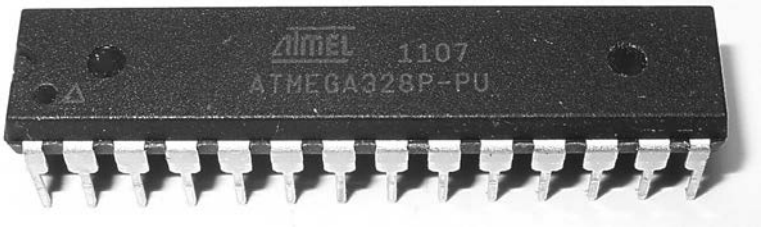


Рис. 2.3. Микроконтроллер

Микроконтроллер — это «мозг» Arduino, маленький выполняющий инструкции компьютер с микропроцессором, имеющий несколько видов памяти для хранения данных и инструкций и имеющий различные входы и выходы для вывода или ввода данных. Чуть ниже микроконтроллера есть две группы контактов, как показано на рис. 2.4.

Рис. 2.4. Контакты питания и аналоговые входы

Слева находятся контакты питания и контакты для подключения внешней кнопки сброса. Группа справа — шесть контактов аналоговых входов. Они используются для измерения уровней напряжения электрических сигналов. Контакты A4 и A5 могут использоваться для обмена данными с другими устройствами.

---

[1] Согласно спецификации рекомендованное напряжение должно находиться в диапазоне 7–12 В. — *Примеч. науч. ред.*

Вдоль верхнего края платы располагаются еще две группы разъемов, как показано на рис. 2.5.

Рис. 2.5. Контакты цифровых входов/выходов

Контакты с номерами от 0 до 13 — это цифровые входы/выходы. С их помощью можно определять наличие или отсутствие входных электрических сигналов или генерировать выходные. Контакты с номерами 0 и 1 известны как *последовательный порт* и могут использоваться для обмена данными с другими устройствами, например с компьютером через преобразователь последовательного интерфейса — USB (UART-USB). Разъемы со знаком тильды (~) могут генерировать электрический сигнал переменного напряжения (который на экране осциллографа выглядит как морские волны, именно поэтому и выбран волнообразный знак тильды). Эти контакты можно использовать, к примеру, для создания световых эффектов или управления электродвигателями.

На плате Arduino есть несколько хорошо знакомых нам устройств — *светодиодов*. Они испускают свет, когда через них течет ток. На плате четыре светодиода: один, с подписью ON, находится возле правого края и служит индикатором подключенного к плате электропитания. Другие три располагаются ближе к левому краю, рядом друг с другом, как показано на рис. 2.6.

Светодиоды с подписями RX и TX зажигаются во время обмена данными между платой Arduino и подключенными устройствами через последовательный порт и USB. Светодиод L предназначен для нужд пользователя (он подключен к контакту цифрового входа/выхода с номером 13). Небольшой черный квадрат слева от светодиодов — это микроконтроллер, управляющий интерфейсом USB. Именно он позволяет Arduino обмениваться данными с компьютером, но вам едва ли придется иметь с ним дело.

И наконец, на рис. 2.7 показана кнопка RESET (Сброс).

Как это иногда случается с компьютерами, в плате Arduino тоже может что-то пойти не так. Когда все ваши попытки «привести ее в чувство» не возымеют успеха, останется только нажать кнопку сброса и перезапустить Arduino. Простая кнопка RESET (Сброс) на плате предназначена для перезапуска Arduino и решения проблем, неустранимых другими способами.

Рис. 2.6. Светодиоды на плате

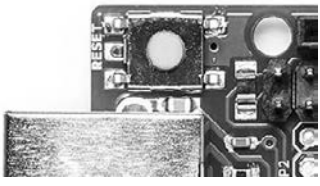


Рис. 2.7. Кнопка RESET (Сброс)

Одна из замечательных особенностей системы Arduino — простота ее расширения. К ней легко добавляются новые аппаратные функции. Два ряда контактов вдоль каждой стороны позволяют подключить *плату расширения* (shield в терминах Arduino) — другую плату с контактами, которые подключаются в разъемы на Arduino. На рис. 2.8 показана плата расширения с интерфейсом Ethernet, с помощью которой Arduino может обмениваться данными через сети и интернет.

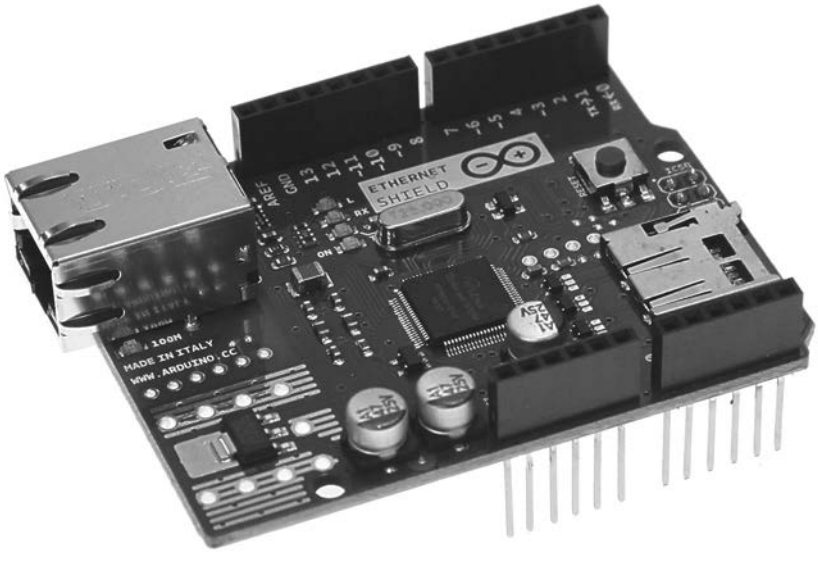


Рис. 2.8. Плата расширения Ethernet для Arduino

Обратите внимание, что у платы расширения Ethernet тоже есть дополнительные гнездовые ряды контактов. Это позволяет подключать сверху дополнительные платы расширений. На рис. 2.9 показана еще одна, вставленная сверху, плата

с большими цифровыми индикаторами, датчиком температуры, дополнительным устройством хранения данных и большим светодиодом.

Помните, что во избежание «конфликтов» вы должны знать назначение контактов платы расширения. В продаже есть макетные платы, на которых можно собирать свои схемы. Об этой возможности мы поговорим в главе 7.

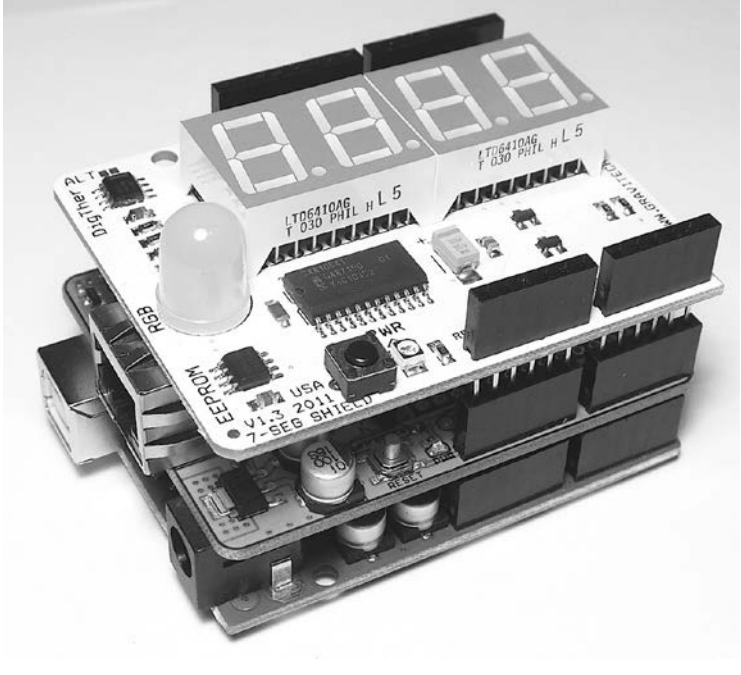


Рис. 2.9. Плата с цифровыми индикаторами и датчиком температуры

Аппаратную часть Arduino дополняет *программная* — набор инструкций, сообщающих аппаратной части, что и как делать.

В главе 1 вы уже установили ПО IDE на свой компьютер и настроили для работы с платой Arduino. Теперь мы поближе познакомимся с IDE и напишем первую простую программу (или *скетч*) для Arduino.

Обзор среды разработки

Из рис. 2.10 мы видим, что среда разработки Arduino IDE напоминает простой текстовый процессор. Окно IDE делится на три основные области: управление, ввод текста и вывод сообщений.

Область управления

Область управления (см. рис. 2.10, *вверху*) содержит заголовок окна, меню и панель инструментов. В заголовке окна отображается имя файла скетча (например, *Blink*) и версия IDE (например, *Arduino 2.0.0-beta.4*). Ниже находится полоса меню (File (Файл), Edit (Правка), Sketch (Скетч), Tools (Инструменты) и Help (Помощь)) и панель инструментов с пиктограммами.

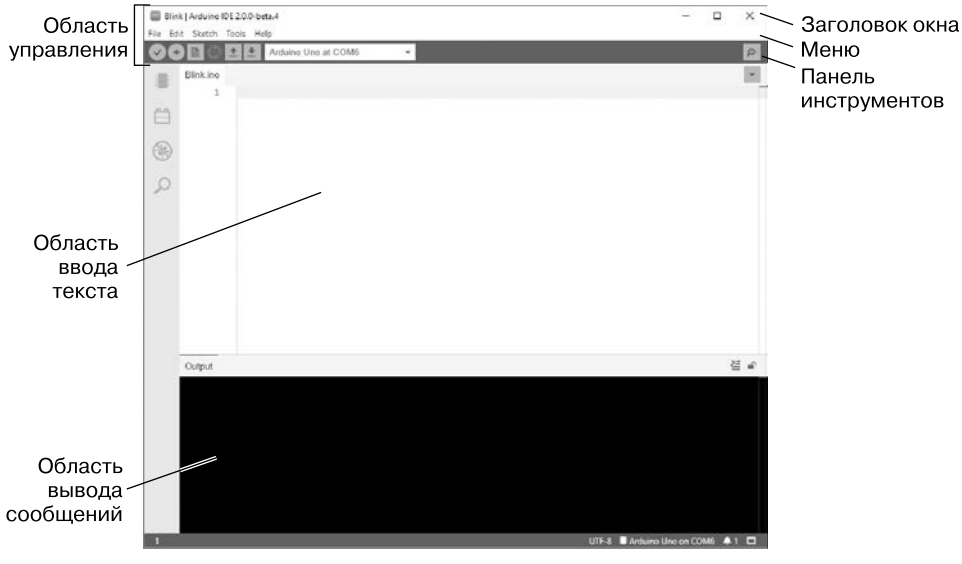


Рис. 2.10. Среда разработки Arduino IDE

Меню

Как и в другом текстовом процессоре или редакторе щелчком на любом элементе меню выводится список соответствующих команд:

- File (Файл) — команды для сохранения, загрузки и печати скетчей; богатый набор примеров скетчей, которые можно открыть; подменю Preferences (Настройки);
- Edit (Правка) — команды копирования, вставки и поиска, обычные для любого текстового процессора;
- Sketch (Скетч) — команда для проверки и компиляции скетча перед загрузкой в плату, команда доступа к папке со скетчем и команда импортирования;
- Tools (Инструменты) — много разных команд и команды для выбора типа платы Arduino и порта USB;
- Help (Помощь) — ссылки на разные темы и версию IDE.

Панель инструментов

Под полосой меню есть панель инструментов с шестью пиктограммами. Если навести указатель мыши на любую из них, появится название соответствующей команды. Ниже кратко описываются пиктограммы в порядке слева направо:

- Verify (Проверить) запускает проверку скетча на корректность и отсутствие программных ошибок;
- Upload (Загрузить) запускает проверку скетча и последующую его загрузку в плату Arduino;
- New (Новый) открывает новый пустой скетч в новом окне;
- Open (Открыть) открывает ранее сохраненный скетч;
- Save (Сохранить) сохраняет открытый скетч. Если у него еще нет названия, вам будет предложено ввести его;
- Serial Monitor (Монитор порта) открывает новое окно для обмена данными между платой Arduino и IDE.

Область ввода текста

Область ввода текста (см. рис. 2.10, *в центре*) служит для ввода исходного кода скетчей. Имя текущего скетча отображается на вкладке вверху слева (по умолчанию скетчу присваивается имя с текущей датой). Здесь вводится исходный код скетча, как в обычном текстовом редакторе.

Область вывода сообщений

Область вывода сообщений (см. рис. 2.10, *внизу*) — черная прямоугольная область в нижней части окна IDE. В ней будут выводиться самые разные сообщения, включая результаты проверки скетчей, успешность загрузки в плату и др.

Справа внизу под областью вывода сообщений отображается тип вашей платы Arduino и порт USB, к которому она подключена, — в данном случае *Arduino Uno on COM6*.

Создание первого скетча в IDE

Скетч для Arduino — это набор инструкций, определяющий пути решения стоящей перед вами задачи. Другими словами, это *программа*. В этом разделе вы создадите и загрузите простой скетч, который заставит мигать светодиод на плате (рис. 2.11), включая и выключая его ежесекундно.

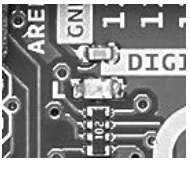


Рис. 2.11. Светодиод с меткой L на плате Arduino

ПРИМЕЧАНИЕ

Не стремитесь досконально разобраться в командах в скетче, представленном здесь. Его цель — показать, насколько просто заставить Arduino работать по заданной вами программе. Поэтому, встретив что-то непонятное, не останавливайтесь и продолжайте читать.

Для начала подключите плату Arduino к компьютеру с помощью кабеля USB. Затем запустите IDE и в раскрывающемся списке выберите вашу плату (Arduino Uno) и тип порта USB, как показано на рис. 2.12. Это позволит убедиться, что плата правильно подключена.

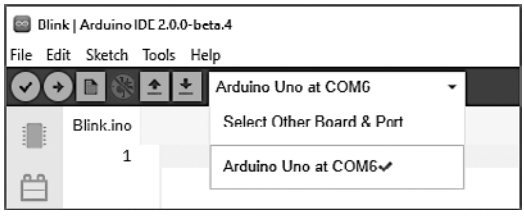


Рис. 2.12. Выбор платы Arduino Uno

Комментарии

Сначала введем комментарий, описывающий назначение скетча. *Комментарий* — это примечание любой длины, находящееся в исходном коде скетча и написанное для пользователя. С помощью комментариев можно оставлять примечания для себя или других, инструкции или описание важных деталей. При создании скетчей для Arduino всегда полезно добавлять комментарии, описывающие ваши намерения. Они пригодятся вам и в том случае, если вы вновь вернетесь к скетчу позднее.

Чтобы добавить однострочный комментарий, введите два символа слеша и текст комментария, например:

```
// Скетч мигает светодиодом. Автор: Мэри Смит, создан 07/01/21
```

Два слеша подряд сообщают среде разработки, что она должна игнорировать следующую за ними строку во время *проверки* скетча на отсутствие ошибок.

Чтобы написать комментарий из нескольких строк, введите символы `/*` в строке перед ним и символы `*/` в строке после него:

```
/*
Скетч мигает светодиодом
Автор: Мэри Смит, создан 07/01/21
*/
```

Комбинации символов `/*` и `*/` сообщают среде разработки, что она должна игнорировать текст между ними.

Оставьте комментарий одним из этих способов и сохраните скетч, выбрав пункт меню File ▸ Save As (Файл ▸ Сохранить как). Введите короткое имя скетча (например, `blinky`) и нажмите OK.

По умолчанию среда разработки сохраняет скетчи в файлах с расширением `.ino` и добавляет его автоматически. В данном случае скетч должен сохраниться в файле с именем `blinky.ino` и появиться в папке с вашими скетчами.

Функция setup()

Следующий этап в создании любого скетча — добавление функции `void setup()`. Она содержит инструкции, которые плата Arduino выполняет только один раз после каждого сброса или включения питания. Чтобы создать функцию `setup()`, добавьте после комментария следующие строки:

```
void setup()
{
}
```

Управление аппаратными компонентами

Наша программа должна включать и выключать светодиод L на плате Arduino. Этот светодиод подключен к контакту цифрового входа/выхода с номером 13. Цифровые входы/выходы либо определяют наличие электрического сигнала, либо генерируют сигнал по команде. В этом проекте мы будем генерировать сигнал, включающий светодиод.

Введите следующую строку между фигурными скобками (`{` и `}`):

```
pinMode(13, OUTPUT); // настроить контакт 13 как цифровой выход
```

Число `13` в листинге определяет нужный нам контакт. Он настраивается на работу в режиме `OUTPUT`, то есть он будет генерировать (выводить — output) электрический сигнал. Если нужно определять наличие входящего электрического сигнала, следует указать режим `INPUT`. Обратите внимание, что вызов функции `pinMode()` завершается точкой с запятой (`;`). Каждая строка с инструкциями в скетчах Arduino должна завершаться именно так.

В конце сохраните скетч еще раз, чтобы случайно не потерять результаты своих трудов.

Функция loop()

Как вы помните, наша цель — заставить светодиод включаться и выключаться снова и снова. Для этого создадим функцию `loop()`, которую плата Arduino будет выполнять, пока кто-то не выключит питание или не нажмет кнопку RESET (Сброс).

Введите код, выделенный полужирным шрифтом в следующем листинге, чтобы создать пустую функцию `loop()`. Не забудьте завершить новый раздел закрывающей фигурной скобкой (`}`) и затем снова сохраните скетч.

```
/*
Скетч мигает светодиодом
Автор: Мэри Смит, создан 07/01/21
*/
void setup()
{
  pinMode(13, OUTPUT); // Настроить контакт 13 как цифровой выход
}
void loop()
{
  // Поместите сюда код, который должен выполняться в цикле
}
```

> **ВНИМАНИЕ**
>
> Среда разработки Arduino IDE не поддерживает автоматическое сохранение скетчей, поэтому старайтесь почаще сохранять свою работу!

Теперь поместите в тело `void loop()` вызовы функций, которые будет выполнять Arduino.

Введите следующий код между фигурными скобками, ограничивающими тело функции `loop()`, а после щелкните на пиктограмме Verify (Проверить), чтобы убедиться, что при вводе не было допущено ошибок:

```
digitalWrite(13, HIGH); // Включить напряжение на выходе 13
delay(1000);            // Пауза в одну секунду
digitalWrite(13, LOW);  // Выключить напряжение на выходе 13
delay(1000);            // Пауза в одну секунду
```

Давайте поближе познакомимся с этими командами. Функция `digitalWrite()` управляет напряжением, которое подается на цифровой выход — в нашем случае на контакт 13, к которому подключен светодиод L. Второй параметр в вызове этой функции (`HIGH`) указывает, что она должна установить «высокий» (high) уровень

напряжения. В результате через контакт и светодиод L потечет электрический ток и светодиод включится.

Функция `delay()` приостанавливает работу скетча на указанный период времени — в данном случае после включения светодиода. `delay(1000)` гарантирует, что он останется включенным следующие 1000 миллисекунд (одну секунду).

Затем мы снимаем напряжение со светодиода вызовом `digitalWrite(13, LOW);`. Течение электрического тока останавливается, и светодиод гаснет. В конце вызовом `delay(1000);` выполняется еще одна пауза в одну секунду, в течение которой светодиод остается выключенным.

Ниже приводится полный исходный код скетча:

```
/*
Скетч мигает светодиодом
Автор: Мэри Смит, создан 07/01/21
*/
void setup()
{
  pinMode(13, OUTPUT); // Настроить контакт 13 как цифровой выход
}
void loop()
{
  digitalWrite(13, HIGH); // Включить напряжение на выходе 13
  delay(1000);            // Пауза в одну секунду
  digitalWrite(13, LOW);  // Выключить напряжение на выходе 13
  delay(1000);            // Пауза в одну секунду
}
```

Прежде чем продолжить, сохраните скетч!

Проверка скетча

Проверка скетча позволяет убедиться, что в нем нет ошибок и он понятен Arduino. Для проверки скетча щелкните на пиктограмме Verify (Проверить) в IDE и немного подождите. Когда проверка скетча закончится, в области вывода сообщений должен появиться текст, как показано на рис. 2.13.

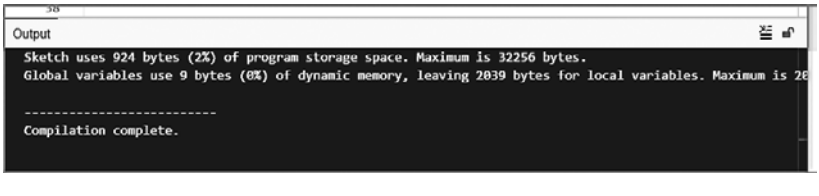


Рис. 2.13. Скетч успешно проверен

Сообщение Done compiling (Компиляция завершена) говорит о том, что скетч проверен и готов к загрузке в плату Arduino. Здесь показано, какой объем памяти будет использован (в данном случае 924 байта) из имеющихся 32 256 байт.

А что, если проверка не увенчалась успехом? Допустим, вы забыли добавить точку с запятой после второго вызова функции `delay(1000)`. Если во время проверки в скетче найдутся ошибки, в области вывода сообщений должно появиться сообщение об ошибке, как показано на рис. 2.14.

Рис. 2.14. Информация об ошибке в области вывода сообщений

IDE выводит текст с описанием ошибки (`expected ';' before '}' token` — «отсутствует точка с запятой перед }») и выделяет в исходном коде строку с ошибкой или строку, следующую за ней. Это помогает быстро найти ошибку и устранить ее.

Загрузка и запуск скетча

Убедившись, что скетч был введен верно, сохраните его, проверьте подключение платы Arduino и щелкните на пиктограмме Upload (Загрузить) на панели инструментов IDE. Среда разработки проверит скетч еще раз и потом загрузит его в плату. В процессе загрузки мигание светодиодов TX/RX на плате (рис. 2.6) подтвердит, что идет обмен информацией между Arduino и компьютером.

После этого наступит момент истины: плата Arduino начнет выполнять ваш скетч. Если все было сделано правильно, светодиод L начнет мигать с секундными интервалами!

Поздравляю. Теперь вы знаете, как ввести, проверить и загрузить скетч Arduino.

Изменение скетча

После запуска скетча у вас может возникнуть желание поменять его поведение, например изменить длительность интервала времени включения/выключения светодиода. IDE во многом похож на текстовый редактор, поэтому вы можете открыть сохраненный скетч, изменить необходимые значения, сохранить его и загрузить в плату. Чтобы увеличить частоту мигания, уменьшите параметры в обоих

вызовах функции `delay` до одной четверти секунды (до 250 миллисекунд), как показано ниже:

```
delay(250); // Пауза в одну четвертую секунды
```

Затем снова загрузите скетч. После этого светодиод начнет мигать чаще, с интервалами в одну четверть секунды.

Что дальше?

Вооруженные новоприобретенными знаниями и умениями ввода, правки, сохранения и загрузки скетчей Arduino, вы готовы перейти к следующей главе. Там вы познакомитесь с другими функциями и принципами проектирования, сконструируете простую электронную схему и узнаете еще много интересного.

3

Первые шаги

В этой главе вы познакомитесь:

- с принципами проектирования;
- основными свойствами электричества;
- компонентами электронных схем: резисторами, светодиодами, транзисторами, выпрямительными диодами и реле;
- макетными платами для навесного монтажа электронных схем;
- целочисленными переменными, циклами `for` и цифровыми выходами —

и узнаете, как с их помощью создавать разные эффекты со светодиодами.

Теперь вам предстоит вдохнуть жизнь в свою плату Arduino. Вы убедитесь, что круг решаемых задач не ограничивается лишь платой, и узнаете, как планировать проекты, чтобы реализовать свои идеи. Потом вы перейдете к краткому учебнику по электричеству. Электричество — движущая сила всего, о чем идет речь в этой книге. Поэтому для создания своих проектов очень важно знать и понимать основы. Еще вы познакомитесь с компонентами электронных схем, которые помогают реализовывать проекты. Помимо этого, вас ждут новые функции — строительные блоки скетчей Arduino.

Планирование проектов

На первых порах у вас может возникнуть соблазн немедленно сесть и написать скетч, чтобы реализовать новую идею. Но перед написанием кода нужно подготовиться. В конце концов, плата Arduino не может читать ваши мысли — ей нужны точные инструкции. И даже если она сможет выполнить их, полученные результаты могут сильно вас удивить, если вы упустите из виду какую-нибудь мелочь.

Что бы вы ни задумали — проект, просто мигающий светодиодом или управляющий моделью железной дороги, в основе успеха всегда лежит подробный план.

1. **Определение цели.** Определите конечный результат вашего проекта.
2. **Разработка алгоритма.** *Алгоритм* — это набор инструкций, описывающих работу проекта. Он должен перечислять шаги для достижения желаемого результата.
3. **Выбор аппаратуры.** Определите, какое дополнительное оборудование будет подключаться к плате Arduino.
4. **Разработка скетча.** Создайте начальную версию программы, которая сообщит плате Arduino, что она должна делать.
5. **Соединение компонентов проекта.** Подключите дополнительные устройства к плате Arduino.
6. **Тестирование и отладка.** Что-то не работает? На этом этапе вы должны выявить ошибки и определить причины их возникновения. Они могут быть в скетче, оборудовании или алгоритме.

Чем больше времени вы потратите на подготовку, тем меньше — на тестирование и отладку.

ПРИМЕЧАНИЕ

Даже детально проработанные проекты иногда становятся жертвой попытки расширения возможностей. Это происходит, если разработчик решает дополнить проект новыми функциями и пытается добавить новые элементы в готовую конструкцию. Если вы захотите изменить проект, не пытайтесь «втиснуть» в него новые элементы или что-то поменять в последний момент. Вместо этого возьмитесь за новый и начните с определения новых целей.

Об электричестве

Давайте теперь немного поговорим об электричестве, ведь совсем скоро вам предстоит конструировать электрические схемы для своих проектов на Arduino. Выражаясь простым языком, *электричество* — это форма энергии, которую можно использовать и преобразовывать в тепло, свет или механическую работу. У него есть три основные важные для нас характеристики: сила тока, напряжение и мощность.

Сила тока

Электрический ток — упорядоченное движение заряженных частиц. Несмотря на то что в большинстве материалов за протекание электрического тока отвечают отрицательно заряженные электроны, в электротехнике принято условно считать,

что ток течет через цепь (путь, проложенный для тока) от положительного полюса источника питания, например батарейки, к отрицательному. Такой ток называют *постоянным* (в книге мы не будем иметь дел с *переменным током*). На некоторых схемах отрицательный полюс обозначается как *«земля»* (ground, GND). Сила тока измеряется в *амперах* (А). Один ампер — это такая сила тока, когда за одну секунду поперечное сечение проводника пересекает $6{,}2415 \times 10^{18}$ электронов. Ток небольшой силы измеряется в *миллиамперах* (мА): 1000 миллиампер равны 1 амперу.

Напряжение

Напряжение — это мера разности потенциалов между положительным и отрицательным полюсами схемы. Измеряется в *вольтах* (В). Если представить, что электроны — это капли в потоке воды, то напряжение будет эквивалентно давлению: чем больше напряжение, тем быстрее ток бежит по цепи.

Мощность

Мощность — это выражение скорости, с которой электрическое устройство преобразует энергию из одной формы в другую. Мощность измеряется в *ваттах* (Вт). Например, лампочка мощностью 100 Вт светит ярче лампочки в 60 Вт, потому что более мощная лампочка преобразует больше электроэнергии в свет и тепло за единицу времени.

Между напряжением, силой тока и мощностью есть простое математическое соотношение:

$$\text{мощность (Вт)} = \text{напряжение (В)} \times \text{ток (А)}.$$

Электронные компоненты

Теперь, когда вы вспомнили все, что знали об электричестве, посмотрим на принцип работы некоторых электронных компонентов и устройств. Электронные *компоненты* — это разные элементы, управляющие электрическим током в схемах. Как и различные компоненты автомобиля обеспечивают хранение, подачу, фильтрацию и впрыск топлива в двигатель, поддерживая его работу, электронные компоненты, управляющие электротоком и использующие его, помогают создавать полезные устройства.

В этой книге я буду описывать специализированные компоненты в той последовательности, в которой они будут появляться в наших проектах. Но о некоторых основных я начну рассказывать прямо сейчас и продолжу в следующих разделах.

Резистор

Для работы некоторых компонентов вроде светодиодов на плате Arduino не нужна большая сила тока — обычно хватает около 10 мА. Когда светодиод получает избыточный ток, он преобразует его в тепловую энергию, большое количество которой может его сжечь. Чтобы уменьшить величину тока на таких компонентах, как светодиоды, между источником напряжения и компонентом можно добавить *резистор*. Ток свободно течет по медным проводникам, но, когда на его пути встречается резистор, величина тока уменьшается. Часть электроэнергии преобразуется в тепло и рассеивается. На рис. 3.1 показана пара типичных резисторов.

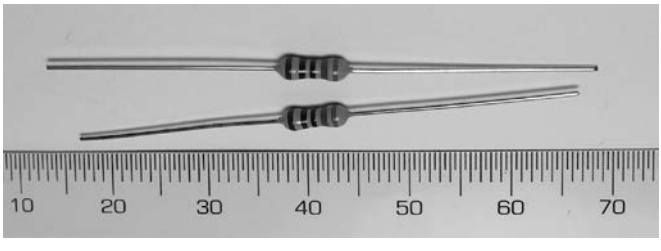


Рис. 3.1. Типичные резисторы

Сопротивление

Сопротивление резистора может быть постоянным или переменным. Оно измеряется в омах (Ом) и может иметь величину от нуля до тысяч ом (*килоом*, или кОм) и даже миллионов ом (*мегаом*, или МОм).

Обозначение номиналов резисторов

Сопротивление резистора можно измерить мультиметром (можно также посмотреть информацию о нем на корпусе резистора). Мы будем работать с очень маленькими резисторами, настолько маленькими, что величину их сопротивления просто невозможно напечатать на корпусе. Поэтому для обозначения сопротивления в таких случаях часто используются цветные полоски, которые читаются слева направо:

- **первая полоса** представляет первую цифру сопротивления;
- **вторая полоса** представляет вторую цифру сопротивления;
- **третья полоса** представляет множитель (для четырехполосных резисторов) или третью цифру сопротивления (для резисторов с пятью полосами);

- **четвертая полоса** представляет множитель для резисторов с пятью полосами или точность — *диапазон отклонений* фактического сопротивления относительно номинала (для резисторов с четырьмя полосами);
- **пятая полоса** определяет диапазон отклонений фактического сопротивления относительно номинала (точность).

В табл. 3.1 перечислены значения цветных полос-маркеров резисторов.

На практике производить резисторы с точным значением сопротивления непросто, поэтому при покупке резисторов вы можете выбрать диапазон отклонений значений в процентах. Для резисторов с пятиполосной маркировкой пятая коричневая полоса соответствует диапазону отклонений 1 %, золотая — 5 % и серебряная — 10 %.

На рис. 3.2 показана схема нанесения цветных полос на корпус резистора. Желтая, фиолетовая и оранжевая соответствуют цифрам 4, 7 и 3, как указано в табл. 3.1. Вместе эти три полосы соответствуют номиналу $47 \times 10^3 = 47\,000$ Ом, или 47 кОм. Последняя, коричневая полоса обозначает погрешность сопротивления, в данном случае 1 %.

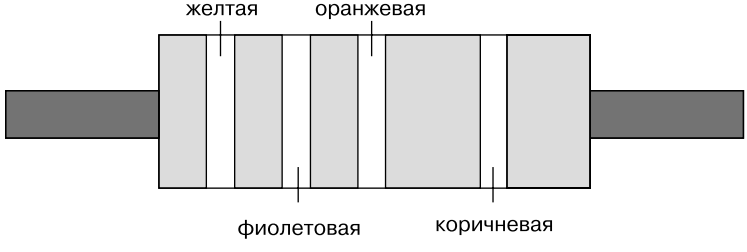


Рис. 3.2. Схема нанесения цветных полос на корпус резистора

Таблица 3.1. Значения цветов полос, используемых для маркировки резисторов

Цвет	Цифра
Черный	0
Коричневый	1
Красный	2
Оранжевый	3
Желтый	4
Зеленый	5
Синий	6
Фиолетовый	7
Серый	8
Белый	9

SMD (чип-резисторы)

На чип-резисторах, используемых для поверхностного монтажа, номинал печатается в виде цифробуквенного кода, как показано на рис. 3.3. Первые две цифры представляют одно число, а третья — количество нулей, следующих за ним. Например, резистор на рис. 3.3 обладает номиналом 10 000 Ом, или 10 кОм.

Рис. 3.3. Чип-резистор для поверхностного монтажа

ПРИМЕЧАНИЕ

Если на поверхности чип-резистора вы увидите цифробуквенное обозначение (например: 01C), поищите в Google по фразе «маркировка EIA-96», чтобы найти таблицы с расшифровкой этой более сложной системы маркировки.

МУЛЬТИМЕТРЫ

Мультиметр — чрезвычайно полезный и относительно недорогой прибор, с помощью которого измеряются напряжение, сопротивление, ток и другие характеристики. На рис. 3.4 изображен мультиметр, с помощью которого измеряется сопротивление резистора.

Рис. 3.4. С помощью мультиметра измеряется сопротивление резистора номиналом 560 Ом и точностью 1 %

Если вы плохо различаете цвета, мультиметр станет вашим основным инструментом. Приобретая этот прибор, выбирайте модель от производителя с надежной репутацией, а не самую дешевую, какую только сможете найти в интернете.

Мощность

Мощность резистора измеряется в ваттах и определяет мощность, которую резистор способен рассеивать, сохраняя работоспособность. Резисторы, изображенные на рис. 3.1, имеют мощность 1/4 Вт и чаще других используются в системах на основе Arduino.

При выборе резистора помните об отношении между мощностью, током и напряжением. Чем выше ток и/или напряжение, тем больше должна быть мощность резистора.

Обычно экземпляры помощнее имеют больший размер. Например, резистор на рис. 3.5 мощностью 5 Вт имеет размеры 22 × 100 мм.

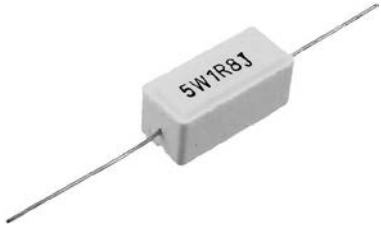


Рис. 3.5. Резистор мощностью 5 Вт

Светодиод

Светодиоды — очень распространенные и бесконечно полезные компоненты, преобразующие электрический ток в свет. Они могут быть разной формы, размера и цвета. На рис. 3.6 показан типичный светодиод.

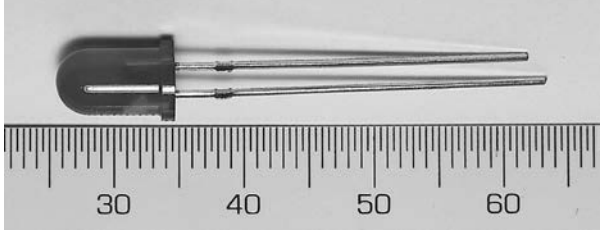


Рис. 3.6. Красный светодиод диаметром 5 мм

Включение светодиода в схему требует определенного внимания, ведь нужно соблюсти *полярность* — ток может входить в светодиод и покидать его только в определенном направлении. Ток должен входить через *анод* (плюс) и покидать светодиод через *катод* (минус), как показано на рис. 3.7. Попытка подать слишком большое напряжение в обратном направлении (равно как и пропустить слишком большой ток в прямом) приведет к выходу светодиода из строя.

К счастью, благодаря конструкции светодиода мы точно можем определить, какой из контактов катод, а какой — анод. Ножка, связанная с анодом, длиннее (для запоминания можно рассуждать так: «плюс» увеличивает длину ножки), а рядом с катодом на бортике корпуса есть плоская площадка, как показано на рис. 3.8.

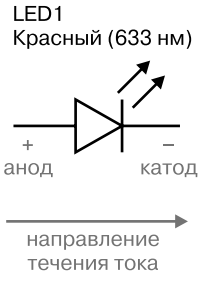


Рис. 3.7. Направление течения тока через светодиод

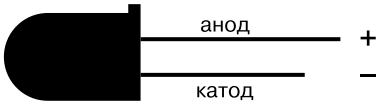


Рис. 3.8. Конструкция светодиода позволяет различить анод (более длинная ножка) и катод (плоская площадка)

Используя светодиоды в проекте, не забывайте о рабочем напряжении и силе тока. Типичным красным светодиодам нужно напряжение около 1,7 В и ток от 5 до 20 мА. Здесь мы сталкиваемся с проблемой, потому что на выходы Arduino подается напряжение 5 В и намного больший ток. К счастью, можно добавить *ограничительный резистор* для уменьшения силы тока, протекающего через светодиод. Но какой номинал резистора выбрать? В этом вам поможет закон Ома.

Вычисление сопротивления ограничительного резистора для светодиода выполняется по следующей формуле:

$$R = (V_s - V_f) / I,$$

где V_s — напряжение питания[1] (Arduino выводит напряжение 5 В); V_f — падение напряжения на светодиоде (например, 1,7 В) и I — сила тока для светодиода (10 мА). Значение I должно выражаться в амперах, то есть 10 мА нужно преобразовать в 0,01 А.

Теперь воспользуемся этой формулой и рассчитаем сопротивление ограничительного резистора для исходных значений $V_s = 5$ В, $V_f = 1{,}7$ В и $I = 0{,}01$ А. Подстановка этих значений в формулу дает величину сопротивления 330 Ом. Но светодиод прекрасно будет работать и при силе тока меньше 10 мА. Считается хорошим тоном по возможности использовать меньшую силу тока, чтобы обеспечить дополнительный уровень защиты схемы. Поэтому для ограничения тока, протекающего через светодиод, мы будем использовать резистор с сопротивлением 560 Ом и мощностью 1/4 Вт, который уменьшит силу тока до 6 мА.

ПРИМЕЧАНИЕ

Если сомневаетесь, всегда выбирайте резистор с большим сопротивлением, потому что тусклый светодиод лучше сгоревшего!

[1] В русскоязычных источниках обычно используется обозначение *U*. — *Примеч. пер.*

ТРЕУГОЛЬНИК ЗАКОНА ОМА

Закон Ома устанавливает отношение между силой тока, сопротивлением и напряжением:

напряжение (*V*) = ток (*I*) × сопротивление (*R*).

Зная две характеристики, можно вычислить третью. Для запоминания закона Ома часто используется треугольная диаграмма, показанная на рис. 3.9.

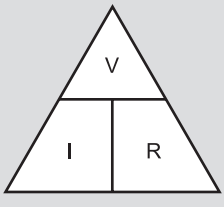


Рис. 3.9. Треугольник закона Ома

Она позволяет быстро вспомнить, как вычислить напряжение, силу тока или сопротивление по двум другим известным значениям. Например, если нужно вычислить сопротивление, закройте пальцем ячейку R, и у вас останется формула, где напряжение делится на силу тока. А чтобы узнать напряжение, закройте ячейку V, и у вас останется формула, в которой сила тока умножается на сопротивление.

Макетная плата для навесного монтажа

Для конструирования электрических схем нужна основа для установки электронных компонентов. С этой целью отлично справляется *макетная плата для навесного (беспаечного) монтажа*. Она имеет пластиковую основу с рядами гнезд, имеющих пружинные клеммы. Платы выпускаются разных размеров, форм и цветов, как показано на рис. 3.10.

Ключ к использованию макетной платы — знание того, как соединены гнезда, — короткие столбцы по вертикали или длинные линии по горизонтали вдоль краев или в центре. На разных платах соединения могут быть выполнены по-разному. Например, на верхней плате из рис. 3.11 пять столбцов гнезд соединены по вертикали, но изолированы по горизонтали. Если подключить два провода к одному вертикальному ряду, они окажутся электрически связанными. Точно так же и два длинных ряда в центре связаны по горизонтали. Поскольку схемы часто нужно подключать к источнику питания, эти длинные горизонтальные ряды гнезд идеально подходят для подачи напряжения.

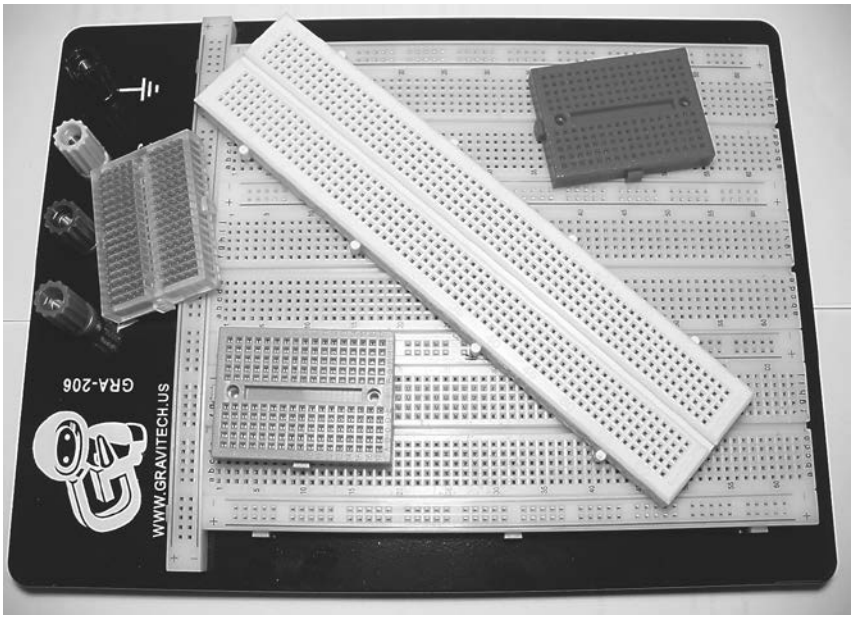


Рис. 3.10. Разные макетные платы

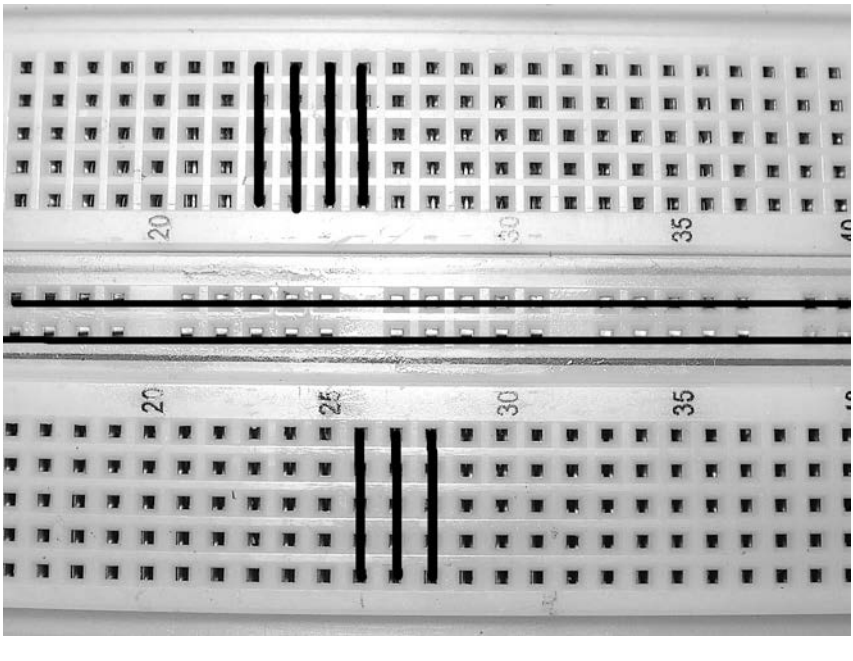


Рис. 3.11. Внутренние соединения на макетной плате

При создании сложных схем не всегда получается разместить компоненты на макетной плате именно там, где хотелось бы. Эта проблема легко решается с помощью отрезков провода. Продавцы макетных плат обычно предлагают и комплекты отрезков провода разной длины, как на рис. 3.12.

Рис. 3.12. Комплект отрезков проводов

Проект 1: бегущая волна из светодиодов

Соберем схему из нескольких светодиодов и резисторов. В этом проекте используется пять светодиодов для имитации эффекта бегущей световой волны на решетке радиатора автомобиля KITT из американского телесериала Knight Rider[1].

[1] В России известен под названием «Рыцарь дорог». — *Примеч. пер.*

Алгоритм

Вот алгоритм работы для этого проекта.

1. Включить светодиод 1.
2. Ждать полсекунды.
3. Выключить светодиод 1.
4. Включить светодиод 2.
5. Ждать полсекунды.
6. Выключить светодиод 2.
7. И так далее до включения светодиода 5. Затем выполнить те же действия в обратном порядке, от светодиода 5 до 1.
8. Повторять предыдущие пункты до бесконечности.

Оборудование

Компоненты для реализации проекта:

- пять светодиодов;
- пять резисторов с номиналом 560 Ом;
- одна макетная плата;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Мы соединим светодиоды с цифровыми выходами со второго по шестой через 560-омные ограничительные резисторы.

Схема

Теперь соберем схему. Ее можно описать несколькими способами. В первых проектах будут использоваться монтажные схемы, как на рис. 3.13.

Сопоставляя монтажную схему с вызовами функций в скетче, можно понять, как она работает. Например, если производится вызов `digitalWrite(2, HIGH)`, на цифровой выход `2` подается ток с напряжением 5 В, который течет через ограничительный резистор, анод светодиода, катод и завершает свой путь в разъеме `GND` («земля»), замыкающем цепь. Дальше, когда происходит вызов `digitalWrite(2, LOW)`, течение тока прекращается и светодиод гаснет.

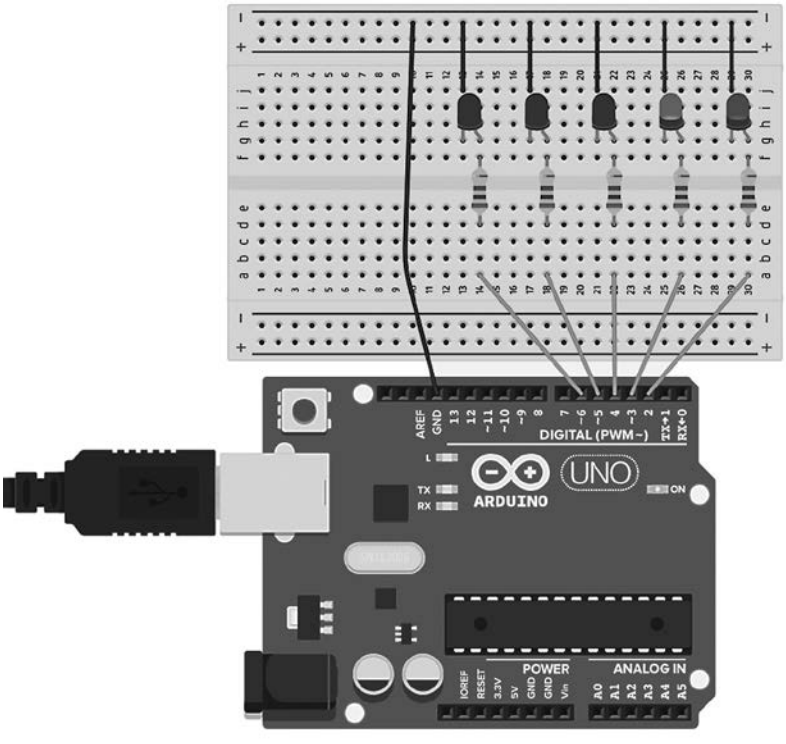


Рис. 3.13. Схема соединений для проекта 1

Скетч

Теперь перейдем к скетчу. Введите следующий код в среде разработки:

```
  // Проект 1 – эффект бегущей волны из светодиодов
❶ void setup()
  {
    pinMode(2, OUTPUT); // Настроить контакты,
    pinMode(3, OUTPUT); // управляющие светодиодами,
    pinMode(4, OUTPUT); // на работу в режиме
    pinMode(5, OUTPUT); // цифровых выходов
    pinMode(6, OUTPUT);
  }

❷ void loop()
  {
    digitalWrite(2, HIGH); // Включить светодиод 1,
    delay(500);            // ждать полсекунды
```

```
    digitalWrite(2, LOW);  // Выключить светодиод 1
    digitalWrite(3, HIGH); // и повторить то же самое
    delay(500);            // для светодиодов со 2-го по 5-й
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    delay(500);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    delay(500);
    digitalWrite(5, LOW);
    digitalWrite(6, HIGH);
    delay(500);
    digitalWrite(6, LOW);
    digitalWrite(5, HIGH);
    delay(500);
    digitalWrite(5, LOW);
    digitalWrite(4, HIGH);
    delay(500);
    digitalWrite(4, LOW);
    digitalWrite(3, HIGH);
    delay(500);
    digitalWrite(3, LOW);
    // В этой точке функция loop() завершится
    // и будет вызвана снова
}
```

В функции `void setup()` ❶ выполняется настройка контактов на работу в режиме цифровых выходов, с их помощью мы будем управлять подачей тока на светодиоды. В функции `void loop()` ❷ мы определяем, когда включить каждый светодиод, вызывая функцию `digitalWrite()`.

Запуск скетча

Теперь подключите Arduino к компьютеру и загрузите скетч. Через одну-две секунды светодиоды должны начать зажигаться и гаснуть по одному, слева направо. Успех окрыляет — прочувствуйте его!

Ну а если вдруг ничего не произошло, отключите USB-кабель от компьютера и проверьте, не ошиблись ли вы при наборе скетча. Если это так, исправьте ошибку и загрузите скетч еще раз. Если в скетче ошибок не обнаружилось, а светодиоды все равно не мигают, проверьте соединения на макетной плате.

Теперь вы знаете, как заставить светодиод мигать с помощью Arduino, но этот скетч не слишком удобный. Если позднее вам захочется заставить волну бежать быстрее, придется вносить изменения в каждый вызов `delay(500)`. Выберем путь получше.

Переменные

Переменные используются в компьютерных программах для хранения данных. Слабое место этого скетча — недостаточная гибкость. Представим, что мы вызывали функцию `delay(500)` для выполнения задержки перед выключением светодиодов. Если позднее вы захотите изменить время задержки, придется вручную менять его везде, где оно встречается в скетче. Для устранения этой проблемы добавим в скетч переменную, представляющую значение задержки для функции `delay()`.

Введите следующую строку в скетч проекта 1 перед функцией `void setup()` сразу за вступительным комментарием:

```
int d = 250;
```

Она присваивает число `250` переменной `d`; `int` указывает, что переменная предназначена для хранения целого числа со знаком (integer) в диапазоне от –32 768 до 32 767. Такой переменной можно присвоить любое целое число.

После замените все числа `500` в скетче именем переменной `d`. Теперь, выполняя скетч, Arduino будет передавать в вызовы функции `delay()` значение переменной `d`. Когда вы загрузите измененный скетч, волна будет бежать существенно быстрее, потому что значение задержки уменьшилось до `250`.

Теперь для изменения задержки достаточно изменить лишь объявление переменной в начале скетча. К примеру, если присвоить переменной число `100`, скорость бегущей волны увеличится еще больше:

```
int d = 100;
```

Поэкспериментируйте. Попробуйте менять величину задержки и последовательность `HIGH` и `LOW`. Но пока не разбирайте схему — она еще пригодится нам в дальнейших проектах из этой главы.

Проект 2: повторение команд с помощью цикла for

В скетчах часто приходится повторять одни и те же команды. Конечно, можно просто скопировать команду в буфер обмена и вставить ее в скетч столько, сколько нужно, но это решение неэффективно и просто будет расходовать память в программах для Arduino. Вместо этого воспользуемся циклом `for`. Его главное достоинство в том, что он позволяет определить, сколько раз нужно выполнить код внутри него.

Чтобы ближе познакомиться с циклом `for`, введите следующий новый скетч:

```
// Проект 2 — повторение команд с помощью цикла for
int d = 100;
```

```
void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
}

void loop()
{
  for ( int a = 2; a < 7 ; a++ )
  {
    digitalWrite(a, HIGH);
    delay(d);
    digitalWrite(a, LOW);
    delay(d);
  }
}
```

Цикл `for` многократно выполняет код в фигурных скобках, пока не будет выполнено некоторое условие. Здесь мы добавили еще одну целочисленную переменную, `a`, которой в начале цикла присвоили `2`. После каждой итерации команда `a++` увеличивает значение этой переменной на 1. Цикл продолжает выполняться снова и снова, пока значение `a` остается меньше 7 (условие). Как только `a` получит значение, равное или больше 7, Arduino двинется дальше и продолжит выполнение кода после цикла `for`.

В цикле `for` можно вести и обратный отсчет итераций, от больших значений к меньшим. Чтобы посмотреть, как это работает, добавьте в скетч проекта 2 следующий код после первого цикла `for`:

```
❶ for ( int a = 5 ; a > 1 ; a-- )
  {
    digitalWrite(a, HIGH);
    delay(d);
    digitalWrite(a, LOW);
    delay(d);
  }
```

Здесь цикл `for` ❶ присваивает переменной `a` начальное значение `5` и потом вычитает из нее 1 после каждой итерации командой `a--`. Этот цикл продолжается, пока значение `a` остается больше 1 (`a > 1`), и завершается, как только `a` станет равным 1 или меньше.

Мы воспроизвели проект 1, используя меньше кода. Загрузите скетч и убедитесь в этом сами!

Изменение яркости светодиода с помощью широтно-импульсной модуляции

Можно не только включать и выключать светодиоды вызовом `digitalWrite()`, но и регулировать яркость их свечения. Это можно сделать, изменяя промежуток времени, когда светодиод находится во включенном и выключенном состоянии, используя *широтно-импульсную модуляцию* (ШИМ — Pulse-Width Modulation, PWM). ШИМ можно использовать для создания иллюзии изменения яркости светодиода, быстро включая и выключая его 500 раз в секунду. Видимая нами яркость определяется отношением интервала времени, когда цифровой выход включен, к интервалу, когда он выключен — когда светодиод светится и не светится. Наш глаз не способен видеть мерцания частотой более 50 раз в секунду, поэтому создается ощущение непрерывного свечения светодиода.

Чем больше *коэффициент заполнения* (отношение времени, когда через контакт течет ток, ко времени, когда ток не течет, в каждом цикле), тем выше воспринимаемая яркость светодиода, подключенного к цифровому выходу.

На рис. 3.14 изображены циклы ШИМ с разными коэффициентами заполнения. Серые области — это периоды, когда светодиод включен. Как видите, длительность периодов свечения светодиода увеличивается с увеличением коэффициента заполнения.

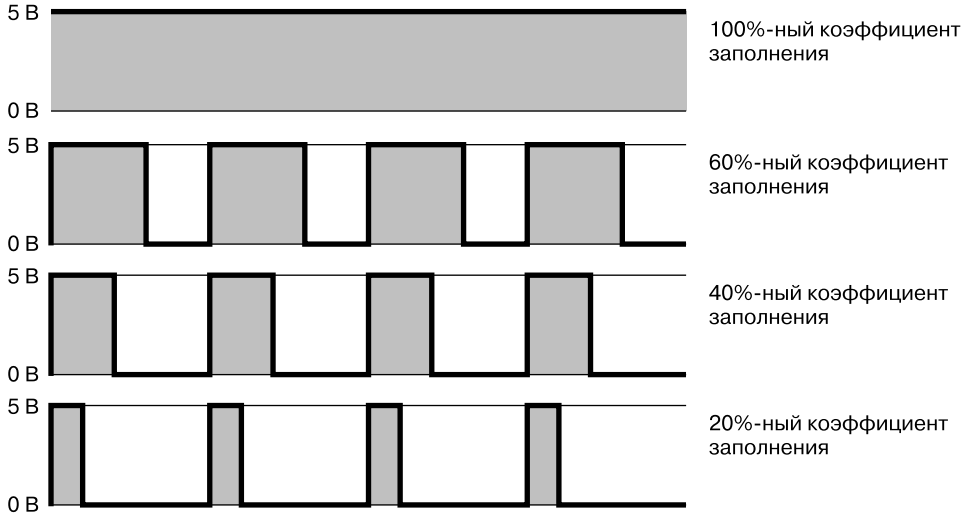


Рис. 3.14. Разные коэффициенты заполнения ШИМ

Только цифровые выходы 3, 5, 6, 9, 10 и 11 на обычной плате Arduino поддерживают ШИМ. Они отмечены символом тильды (~), как показано на рис. 3.15.

Рис. 3.15. Выходы с поддержкой ШИМ отмечены на плате символом тильды (~)

Для создания сигнала ШИМ используется функция `analogWrite(x, y)`, где `x` — номер цифрового выхода, а `y` — значение коэффициента заполнения в диапазоне от 0 до 255, где 0 равен коэффициенту заполнения 0 %, а 255 — 100 %.

Проект 3: демонстрация ШИМ

Теперь используем сигналы ШИМ в нашей схеме из проекта 2. Введите в IDE следующий скетч и загрузите его в плату Arduino:

```
// Проект 3 – демонстрация ШИМ
int d = 5;
void setup()
{
  // настроить контакт 3, управляющий светодиодом и поддерживающий ШИМ,
  // на работу в режиме цифрового выхода
  pinMode(3, OUTPUT);
}

void loop()
{
  for ( int a = 0 ; a < 256 ; a++ )
  {
    analogWrite(3, a);
    delay(d);
  }
  for ( int a = 255 ; a >= 0 ; a-- )
  {
    analogWrite(3, a);
    delay(d);
  }
  delay(200);
}
```

Яркость светодиода, подключенного к цифровому выходу 3, будет плавно уменьшаться и увеличиваться с увеличением и уменьшением коэффициента заполнения.

Иначе говоря, яркость светодиода будет плавно увеличиваться до достижения максимума, а потом плавно уменьшаться. Поэкспериментируйте со скетчем и схемой. Попробуйте заставить все пять светодиодов изменять яркость одновременно или последовательно.

Дополнительные электронные компоненты

Поверьте мне, проекты проще планировать, когда не нужно заботиться о величине тока, потребляемого устройствами под управлением цифровых выходов. При создании проектов помните, что каждый цифровой выход на плате Arduino Uno может отдавать ток не более 40 мА тока на вывод и 200 мА в сумме для всех выводов. Расширить технические возможности Arduino вам помогут три электронных компонента. Их мы и рассмотрим ниже.

ВНИМАНИЕ

Попытка превысить допустимую силу тока в 40 мА на один цифровой выход или 200 мА для всех выходов может вывести из строя микроконтроллер.

Транзистор

Почти все мы знаем слово «транзистор», но большинство не понимает, как он работает. Я постараюсь объяснить это настолько просто, насколько это возможно. Транзистор может включать и выключать намного более мощный ток, чем плата Arduino Uno. При этом мы можем без опаски управлять транзистором с помощью цифровых выходов Arduino. Типичный пример — транзистор BC548 на рис. 3.16.

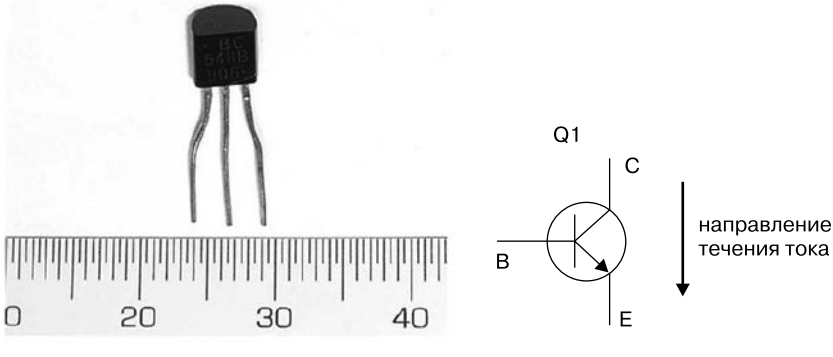


Рис. 3.16. Типичный транзистор: BC548

Как и у светодиодов, у выводов транзистора есть свои уникальные функции, а сами выводы должны подключаться в правильной последовательности. Если смотреть на транзистор с плоской стороны (как показано на рис. 3.16, *слева*), выводы BC548 называются (слева направо) «коллектор», «база» и «эмиттер» (заметьте, что здесь приводится порядок следования выводов, или «цоколёвка», для транзистора BC548 — у других может быть иная схема расположения выводов). Если на базу подается маленький ток, например с цифрового выхода Arduino, то нужный нам больший ток будет протекать через цепь «коллектор — эмиттер». Когда малый управляющий ток снимается с базы, ток через транзистор больше не течет.

Транзистор BC548 может переключать ток силой до 100 мА при напряжении до 30 В — это намного больше, чем способна отдать плата Arduino через цифровой выход. В следующих проектах мы используем этот и другие транзисторы, тогда и познакомимся с ними поближе.

ПРИМЕЧАНИЕ

Всегда обращайте внимание на расположение выводов конкретного транзистора, потому что у каждого может быть своя цоколёвка.

Выпрямительный диод

Диод — очень простой и полезный компонент, позволяющий току течь только в одном направлении. Своей формой он напоминает резистор (рис. 3.17).

Рис. 3.17. Выпрямительный диод 1N4004

В проектах книги мы используем выпрямительный диод 1N4004. Ток протекает через диод от анода к катоду, который обозначен кольцевой меткой на корпусе диода. Такие диоды могут защищать части схем от обратного тока, но это влечет за собой падение напряжения примерно на 0,7 В. Диод 1N4004 способен пропускать ток силой до 1 А и выдерживать обратное напряжение до 400 В, что с лихвой покрывает наши запросы. Это надежный, распространенный и недорогой компонент.

Реле

Реле используются с той же целью, что и транзисторы — для управления током большей силы и напряжения. Преимущество реле состоит в *электрической развязке* от управляющей схемы. Это позволяет плате Arduino переключать токи, оставаясь изолированной и защищенной от их негативного воздействия на плату. Внутри реле есть пара интересных элементов: контакты механического переключателя и низковольтная катушка (рис. 3.18).

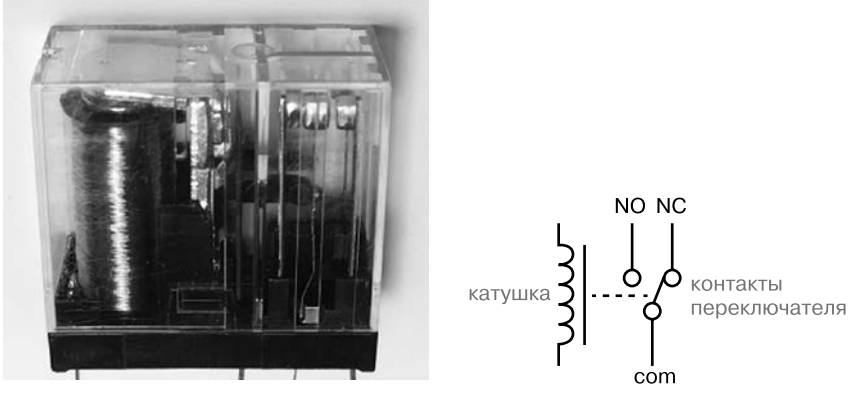


Рис. 3.18. Типичное реле

Когда на катушку подается ток, она начинает действовать как электромагнит и притягивает металлический язычок, работающий как переключатель. Когда электромагнит включен, язычок притягивается к нему и замыкает контакты, когда выключен — возвращается в исходное положение и размыкает их. Так можно управлять контактами, подавая напряжение на катушку и снимая его. Реле включается и выключается с характерным щелчком, который можно услышать при включении поворотника в старом автомобиле.

Высоковольтные схемы

Теперь, после знакомства с транзистором, выпрямительным диодом и реле, используем их для управления током большей силы и напряжения. С их помощью можно управлять включением и выключением мощного электромотора, например. Схема соединения компонентов очень простая (рис. 3.19).

Эта схема управляет реле с 12-вольтовой катушкой. Так можно управлять лампой или вентилятором, подключив их к контактам реле. Цифровой выход 10 на Arduino подключен к базе транзистора через резистор номиналом 1 кОм. Транзистор управляет током, протекающим через катушку, включая и выключая реле. Не забывайте,

что выводы этого транзистора расположены в порядке (слева направо): коллектор — база — эмиттер, со стороны плоской поверхности транзистора. Компонент ❶ слева на макетной плате — это источник питания напряжением 12 В для катушки реле. Отрицательный вывод этого источника питания ❷ подключен к эмиттеру транзистора и контакту GND на плате Arduino. Наконец, катод выпрямительного диода 1N4004 ❸ связан через катушку реле с положительным выводом источника питания. Загляните в документацию с описанием реле, чтобы определить, какие выводы соединены с управляющими контактами, и подключите к ним устройство.

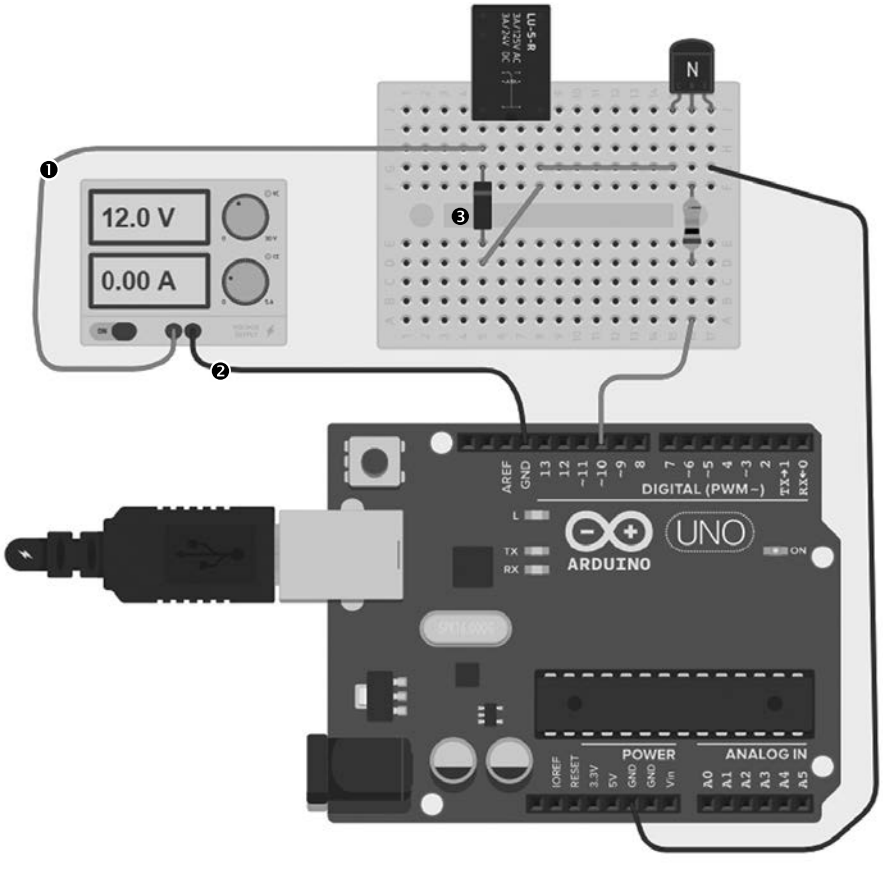


Рис. 3.19. Схема управления реле

Диод здесь служит для защиты схемы. После снятия напряжения с катушки возникает кратковременный всплеск паразитного тока высокого напряжения, который должен быть куда-то направлен. Диод пропускает этот ток по кругу через катушку,

пока не рассеет его в виде тепла. Это предотвращает повреждение транзистора или выхода на плате Arduino от обратного выброса тока.

ВНИМАНИЕ

Если вы захотите с помощью реле управлять приборами, питающимися от бытовой электросети (110–250 В), посоветуйтесь со специалистом. Даже небольшая ошибка может привести к трагическим последствиям.

Что дальше?

Вот и третья глава подошла к концу. Я надеюсь, вам понравились эксперименты со светодиодами. В этой главе мы разными способами создали эффект бегущей световой волны. Вы узнали, как эффективнее использовать функции и циклы для управления компонентами, подключенными к Arduino. И теперь, с полученными знаниями, готовы преодолеть путь к вершинам.

В главе 4 вас ждет много интересного. Вы создадите несколько практических проектов: светофор, термометр, тестер для батареи и многие другие. Итак, если вы готовы подняться на следующий уровень, переверните страницу!

4

Строительные блоки

В этой главе вы:

- научитесь читать принципиальные схемы — язык описания электронных конструкций;
- познакомитесь с конденсатором;
- изучите работу цифровых входов;
- узнаете, как использовать арифметические и логические выражения;
- освоите приемы принятия решений с помощью инструкции `if`;
- узнаете, чем отличаются аналоговые и цифровые сигналы;
- научитесь измерять напряжение на аналоговых входах с разной степенью точности;
- познакомитесь с переменными резисторами, пьезозуммерами и датчиками температуры;
- используете полученные знания для создания светофора, тестера для батареек и термометра.

Эта глава познакомит вас с возможностями платы Arduino. Мы продолжим изучение электроники, в том числе приемов чтения принципиальных схем (путеводителей по электронным конструкциям). Мы исследуем новые компоненты и типы измеряемых сигналов. Дальше мы обсудим дополнительные возможности Arduino — хранение значений, выполнение математических операций и принятие решений. В заключение мы исследуем дополнительные компоненты и на их основе создадим несколько интересных проектов.

Принципиальные схемы

В главе 3 мы научились собирать электронные устройства с помощью монтажных схем, на которых изображены макетная плата и компоненты на ней. Такой способ может показаться самым простым для представления электронных схем. Но чем больше компонентов будет изображено на схеме, тем запутанней она будет. Так как наши проекты со временем будут усложняться, мы перейдем к использованию *принципиальных схем* (также известных как *электрические схемы*). На рис. 4.1 приводится пример такой схемы.

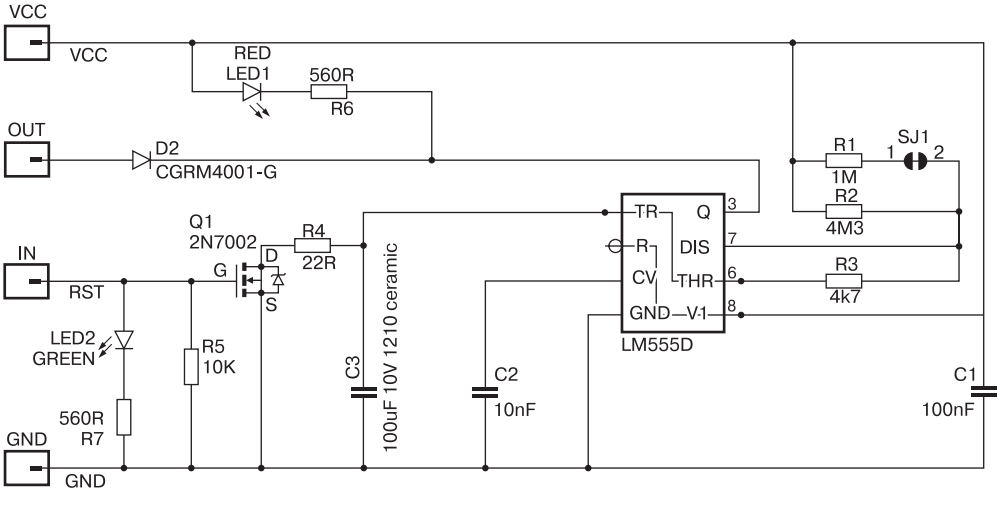


Рис. 4.1. Пример принципиальной схемы

Принципиальные схемы можно назвать путеводителями, показывающими, как электрический ток проходит через разные компоненты. Вместо изображений в схемах используются линии и значки.

Обозначение компонентов

Зная, что символ обозначает, вы легко сможете читать принципиальные схемы. Для начала давайте познакомимся с символами компонентов, которые мы уже использовали.

Arduino

На рис. 4.2 показано, как на схемах обозначается сама плата Arduino. Здесь отмечены и аккуратно подписаны все контакты.

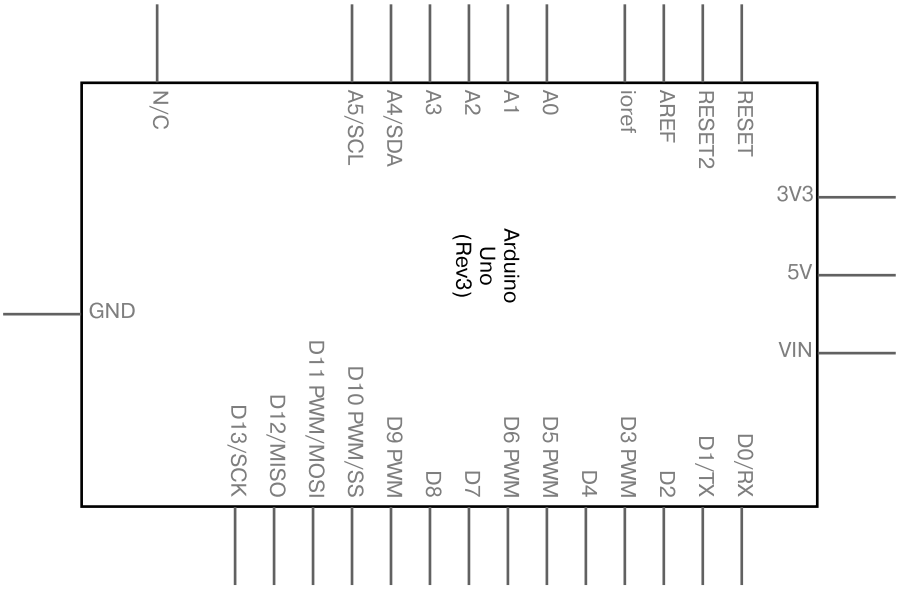


Рис. 4.2. Обозначение платы Arduino Uno

Резистор

На рис. 4.3 показан символ, обозначающий резистор.

На схемах рядом с символом резистора принято изображать его номинал и позиционное обозначение в пределах схемы (в данном случае 220Ω и R1). Это упрощает чтение и анализ принципиальной схемы. Часто номинал в омах обозначается не символом Ω, а буквой R, например: 220 R.

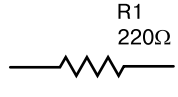


Рис. 4.3. Обозначение резистора

Выпрямительный диод

На рис. 4.4 показан символ, обозначающий выпрямительный диод.

Из главы 3 мы знаем, что выпрямительный диод имеет полярность и пропускает электрический ток от анода к катоду. На рис. 4.4 анод изображен слева, а катод — справа. Чтобы было проще запомнить, представьте, что треугольник — это воронка, в которую втекает электрический ток. Ток не может течь в обратном направлении, потому что вертикальная черта — как «заслонка» — не пускает его.

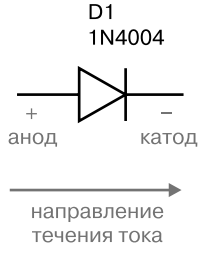


Рис. 4.4. Обозначение выпрямительного диода

Светодиод

На рис. 4.5 показан символ, обозначающий светодиод.

Для обозначения всех компонентов из семейства диодов используется один и тот же символ — треугольник с вертикальной чертой. Но знак светодиода включает еще и две параллельные стрелки, направленные в сторону от него. Они изображают испускаемый свет.

Транзистор

На рис. 4.6 показан символ, обозначающий транзистор. Дальше он будет использоваться для обозначения транзистора BC548.

Вертикальная линия в его верхней части (с буквой C) обозначает коллектор (collector), горизонтальная слева (с буквой B) — базу (base) и вертикальная линия в нижней части (с буквой E) — эмиттер (emitter). Стрелка внутри символа, направленная вправо вниз, сообщает, что этот транзистор относится к типу NPN-транзисторов. Их основная отличительная черта — направление течения электрического тока от коллектора к эмиттеру (другой тип транзисторов — PNP-транзисторы — пропускает электрический ток от эмиттера к коллектору).

Для обозначения транзисторов на схемах используется буква Q, а для резисторов — буква R.

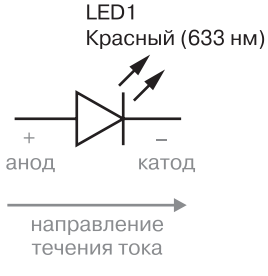


Рис. 4.5. Обозначение светодиода

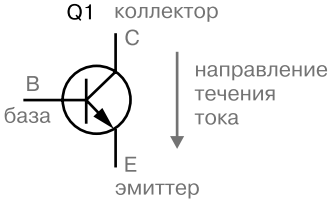


Рис. 4.6. Обозначение транзистора

Реле

На рис. 4.7 показан символ, обозначающий реле.

Реле на схемах отображаются по-разному и могут иметь несколько групп контактов. Но у всех обозначений есть несколько общих элементов. Первый — это представление *катушки* в виде прямой линии со спиралью слева. Второй элемент — это *контакты* реле. Входной контакт часто обозначается как COM (common — «общий»),

а выходные — NO (normally open — «нормально разомкнутый») и NC (normally closed — «нормально замкнутый»).

Символ реле всегда изображается в выключенном состоянии, когда ток *не течет* через катушку — то есть с перемычкой между контактами COM и NC. При подаче напряжения на катушку реле замыкает контакты COM и NO.

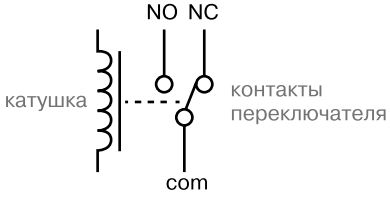


Рис. 4.7. Обозначение реле

Проводники на схемах

Пересекающиеся и соединяющиеся проводники изображаются на схемах по-разному, это видно в следующих примерах.

Пересечение проводников

Пересечение двух проводников без образования электрического соединения может быть обозначено двумя способами (рис. 4.8). В нашем случае предписанного способа обозначения нет, поэтому можно выбрать, что понравится.

Электрическое соединение проводников

Когда подразумевается физическое соединение проводников, в месте пересечения изображается точка соединения, как показано на рис. 4.9.

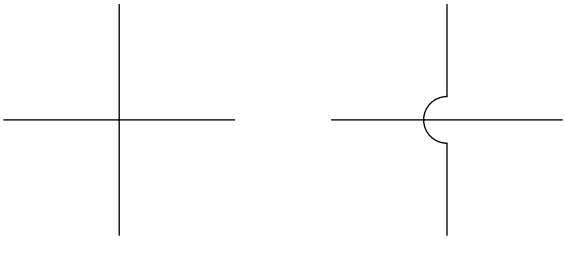


Рис. 4.8. Пересечение проводников

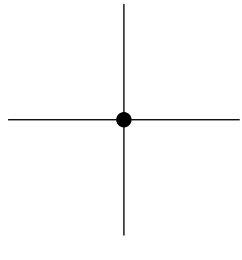


Рис. 4.9. Электрическое соединение проводников

Проводник, подключенный к «земле»

Для обозначения связи проводника с «землей» (GND) принято использовать стандартный символ, изображенный на рис. 4.10.

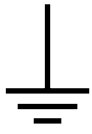


Рис. 4.10. Символ, обозначающий связь с «землей»

Символ «земли» в конце линии на схеме сообщает, что проводник физически подключен к контакту GND на плате Arduino.

Чтение принципиальных схем

Теперь, когда вы ознакомились с основными обозначениями, попробуем прочитать схему, нарисованную для проекта 1. Напомню, что там мы воспроизвели эффект световой волны на пяти светодиодах.

Сравните схемы на рис. 4.11 и рис. 3.13. Согласитесь, что принципиальная схема позволяет намного проще описать конструкцию устройства.

С этого момента в описаниях проектов будут использоваться принципиальные схемы, а обозначения новых компонентов я буду объяснять по мере знакомства с ними.

ПРИМЕЧАНИЕ

При желании вы можете рисовать собственные принципиальные схемы на компьютере. Попробуйте воспользоваться приложением Fritzing, его бесплатно можно скачать по адресу http://www.fritzing.org/.

Конденсатор

Конденсатор — устройство, способное сохранять электрический заряд. Он состоит из двух металлических пластин, разделенных слоем диэлектрика, позволяющего накапливать электрический заряд между пластинами. После отключения конденсатора от электросети заряд на нем остается, но может и «стекать» (в этом случае говорят, что конденсатор *разряжается*), если найдет новый путь для дальнейшего движения.

Измерение емкости конденсатора

Величина электрического заряда, которая может накапливаться в конденсаторе, определяется его *емкостью*. Последняя измеряется в *фарадах*. Один фарад — очень большая величина, поэтому обычно емкость конденсаторов измеряется в пикофарадах или микрофарадах.

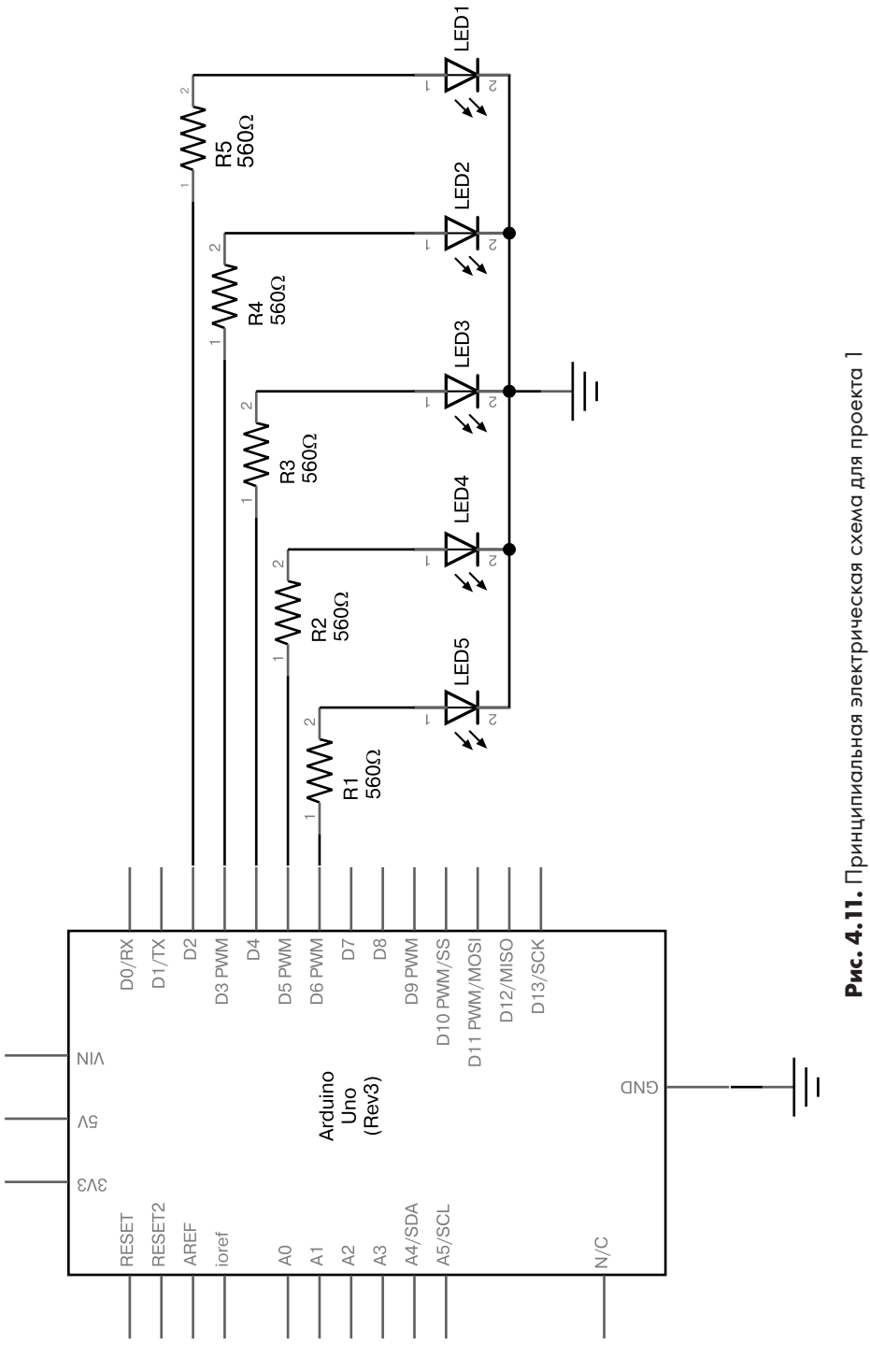


Рис. 4.11. Принципиальная электрическая схема для проекта 1

Один *пикофарад* (пФ, или pF) — это одна триллионная доля фарада, а один *микрофарад* (мкФ, или μF) — одна миллионная. При маркировке конденсаторов указывается и максимальное напряжение, которое способен выдерживать конденсатор. В книге мы будем работать только с низковольтными схемами, поэтому нам не нужны конденсаторы, рассчитанные на напряжение выше 10 В. Но вообще в электронных устройствах принято использовать конденсаторы, рассчитанные на напряжение выше, чем в схеме. Наиболее типичными номинальными напряжениями являются 10, 16, 25 и 50 В.

Маркировка конденсаторов

Для чтения маркировки керамических конденсаторов нужна практика, потому что номинальная емкость печатается на корпусе в виде кода. Первые две цифры — это значение в пикофарадах, а третья определяет множитель в виде числа нулей. К примеру, на конденсатор на рис. 4.12 нанесен код 104. Он означает число 10, за которым следуют четыре нуля. В результате получается 100 000 пикофарад (пФ), 100 нанофарад (нФ) или 0,1 микрофарада (мкФ).

Рис. 4.12. Керамический конденсатор емкостью 0,1 мкФ

ПРИМЕЧАНИЕ

При возникновении сложностей с преобразованием единиц измерения обращайтесь к превосходной таблице по адресу http://www.justradios.com/uFnFpF.html[1].

Типы конденсаторов

В наших проектах будут использоваться конденсаторы двух типов: керамические и электролитические.

Керамические конденсаторы

Керамические конденсаторы (рис. 4.12) отличаются очень маленькими размерами и небольшой емкостью. Они не имеют полярности и могут включаться в схему в любом положении. На схемах неполярный конденсатор изображается, как показано на рис. 4.13.

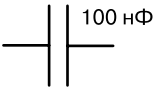


Рис. 4.13. Символ неполярного конденсатора с обозначением емкости справа вверху

[1] Отличный калькулятор перевода величин на русском языке можно найти по адресу http://www.translatorscafe.com/cafe/RU/units-converter/electrostatic-capacitance/. — *Примеч. пер.*

Керамические конденсаторы прекрасно работают в высокочастотных цепях. Благодаря малой емкости они способны очень быстро заряжаться и разряжаться.

Электролитические конденсаторы

Электролитические конденсаторы (рис. 4.14) больше керамических по размеру, обладают большей емкостью, и их выводы различаются полярностью. На корпусе отмечается либо положительный (+), либо отрицательный (–) вывод. На рис. 4.14 можно видеть полосу и маленький знак минус (–), отмечающий отрицательный вывод. Так же как резисторы, конденсаторы имеют диапазон отклонений значений относительно номинала. Конденсатор на рис. 4.14 имеет диапазон отклонений 20 % и емкость 100 мкФ.

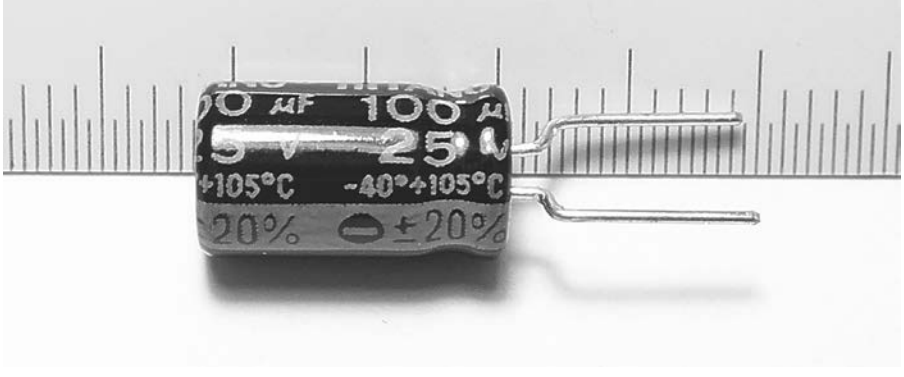


Рис. 4.14. Электролитический конденсатор

На схемах электролитический конденсатор изображается, как показано на рис. 4.15. Знак + обозначает полярность конденсатора.

Электролитические конденсаторы часто используются для накопления больших зарядов и сглаживания пульсаций напряжения. Подобно маленьким батарейкам, они могут демпфировать скачки напряжения и стабилизировать его в цепях, где потребляемый ток быстро меняется. Номинал конденсаторов, к счастью, печатается на корпусе в понятной форме и расшифровывать его не нужно.

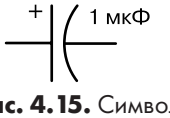


Рис. 4.15. Символ полярного конденсатора

Теперь, когда у вас уже есть опыт вывода простых сигналов с платы Arduino, пришло время научиться вводить сигналы извне и на их основе принимать решения.

Цифровые входы

В главе 3 мы использовали выводы цифровых входов/выходов в качестве выходов для включения и выключения светодиодов. Те же самые контакты можно использовать и в качестве входов, принимающих сигналы от пользователя, при условии что они могут иметь только два состояния — высокое и низкое.

Простейшим источником входного цифрового сигнала может быть *кнопка без фиксации* (рис. 4.16). Ее можно установить прямо на макетной плате и подключить к Arduino. При нажатии кнопки электрический ток потечет через нее на контакт цифрового входа, который определит наличие напряжения.

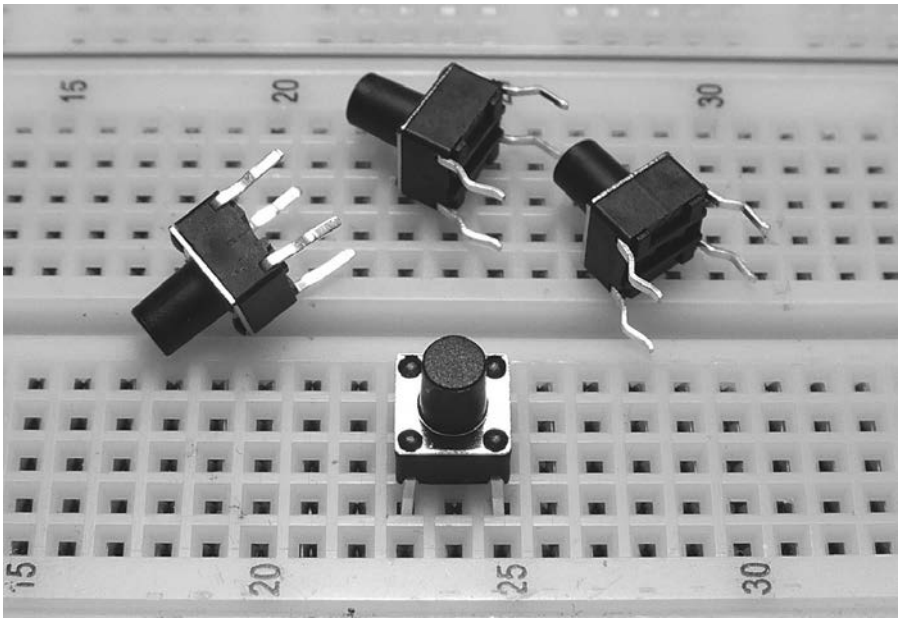


Рис. 4.16. Простые кнопки без фиксации на макетной плате

Обратите внимание, как контакты кнопки на рис. 4.16 вставлены в гнезда на макетной плате и соединяют ряды 23 и 25. Когда кнопка нажата, эти два ряда соединяются. На принципиальных схемах эта конкретная кнопка изображается, как показано на рис. 4.17. На символе видны две стороны кнопки, она обозначается буквенно-цифровым кодом с префиксом S. Когда кнопка нажата, линия замыкает две половины и позволяет электрическому току течь через контакты.

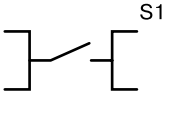


Рис. 4.17. Обозначение кнопки без фиксации

ИССЛЕДОВАНИЕ ЭФФЕКТА ДРЕБЕЗГА КОНТАКТОВ С ПОМОЩЬЮ ОСЦИЛЛОГРАФА С ЦИФРОВОЙ ПАМЯТЬЮ

Кнопки без фиксации имеют эффект под названием «дребезг контактов», или просто «дребезг». Он проявляется в виде череды импульсов, сопровождающей единственное нажатие кнопки. Этот эффект возникает из-за того, что замыкаемые металлические контакты внутри кнопки настолько малы, что могут вибрировать после ее отпускания и очень быстро замыкаться и размыкаться несколько раз подряд.

Эффект дребезга контактов можно наблюдать с помощью *осциллографа с цифровой памятью*. Это устройство отображает изменения напряжения за некоторый период времени. Взгляните на рис. 4.18, где эффект дребезга показан на экране осциллографа.

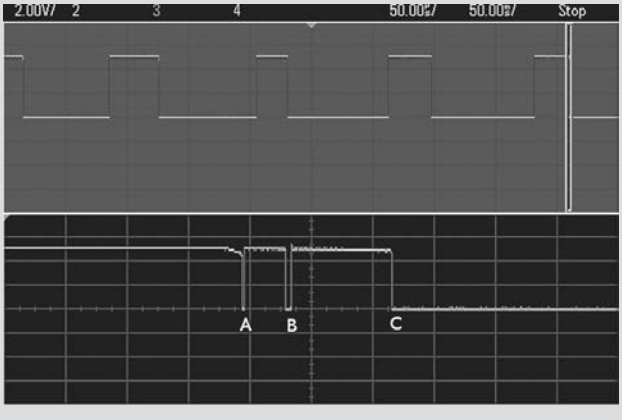


Рис. 4.18. Проявление эффекта дребезга контактов

В верхней половине рис. 4.18 видны результаты нескольких нажатий кнопки. Когда линия напряжения, отмеченная на рис. 4.18 стрелкой, находится в верхнем положении (5 В), кнопка находится в состоянии «включено» и ток проходит через нее. Под словом Stop двумя вертикальными линиями выделен отрезок времени сразу после отпускания кнопки.

На нижней половине рис. 4.18 этот отрезок виден в увеличенном масштабе. В точке A пользователь отпустил кнопку, и уровень напряжения упал до 0 В. Но дальше из-за физических колебаний контактов напряжение снова подскочило до 5 В и оставалось высоким до точки B, где контакты разомкнулись. После этого из-за вибрации контакты вновь замкнулись и разомкнулись уже в точке C. Дальше кнопка перешла в устойчивое состояние «выключено». В результате вместо одного сигнала нажатия кнопки мы послали три.

Проект 4: демонстрация работы цифрового входа

Цель проекта — включить светодиод на полсекунды в ответ при нажатии кнопки.

Алгоритм

Вот алгоритм работы этого проекта.

1. Проверить, нажата ли кнопка.
2. Если кнопка нажата, включить светодиод на полсекунды и затем выключить его.
3. Если кнопка не нажата, ничего не делать.
4. Повторять до бесконечности.

Оборудование

Компоненты для реализации проекта:

- одна кнопка;
- один светодиод;
- один резистор номиналом 560 Ом;
- один резистор номиналом 10 кОм;
- один конденсатор емкостью 100 нФ;
- несколько отрезков провода разной длины;
- одна макетная плата;
- плата Arduino и кабель USB.

Схема

Сначала соберем на макетной плате схему с рис. 4.19. Обратите внимание, что резистор номиналом 10 кОм соединяет вывод GND и контакт 7. Этот резистор называется *подтягивающим вниз*, потому что в отсутствие сигнала он понижает напряжение на цифровом входе почти до нуля. Кроме того, дополнительный конденсатор емкостью 100 нФ, включенный параллельно резистору номиналом 10 кОм, создает простую цепь, устраняющую эффект дребезга контактов, помогая отфильтровать паразитные импульсы. В момент нажатия кнопки на цифровой вход подается напряжение высокого уровня. Когда кнопка отпускается, цифровой

вход «подтягивается к земле» резистором 10 кОм, а конденсатор емкостью 100 нФ создает небольшую задержку. Величина этой задержки дольше, чем продолжительность дребезга контактов кнопки. Задержка замедляет падение напряжения до 0 В, устраняя большинство ложных срабатываний.

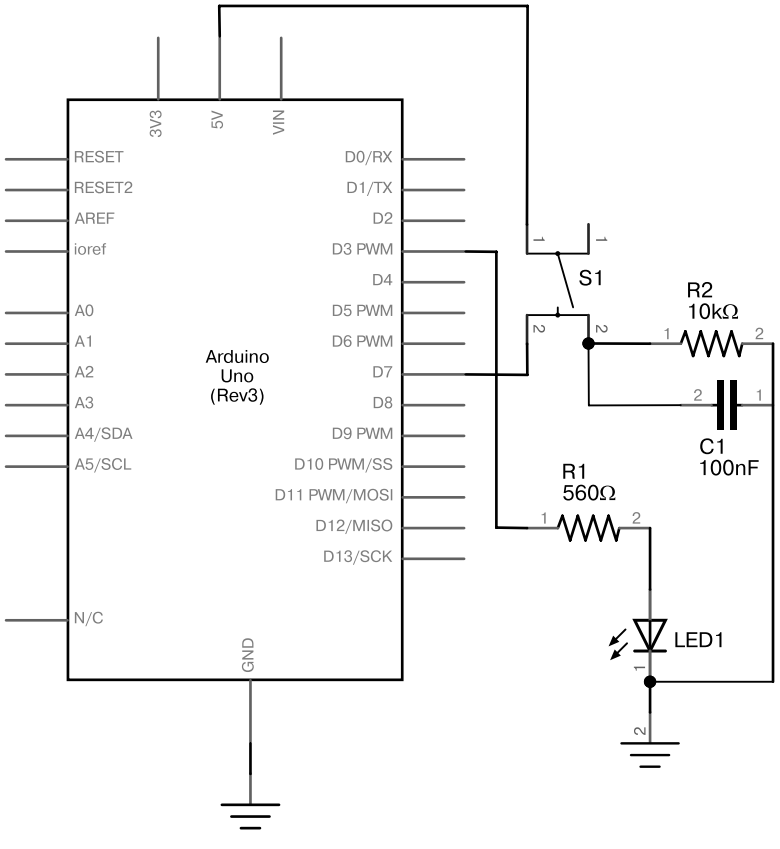


Рис. 4.19. Принципиальная схема проекта 4

Поскольку это первый проект, который собирается с применением принципиальной схемы, ниже есть пошаговая инструкция по ее сборке. Это поможет понять, как соединять компоненты.

1. Вставьте кнопку в макетную плату, как показано на рис. 4.20.
2. Вставьте резистор номиналом 10 кОм, короткий проводник и конденсатор, как показано на рис. 4.21.

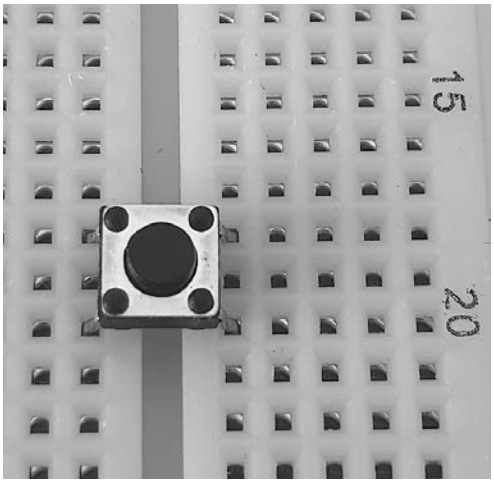


Рис. 4.20. Кнопка на макетной плате

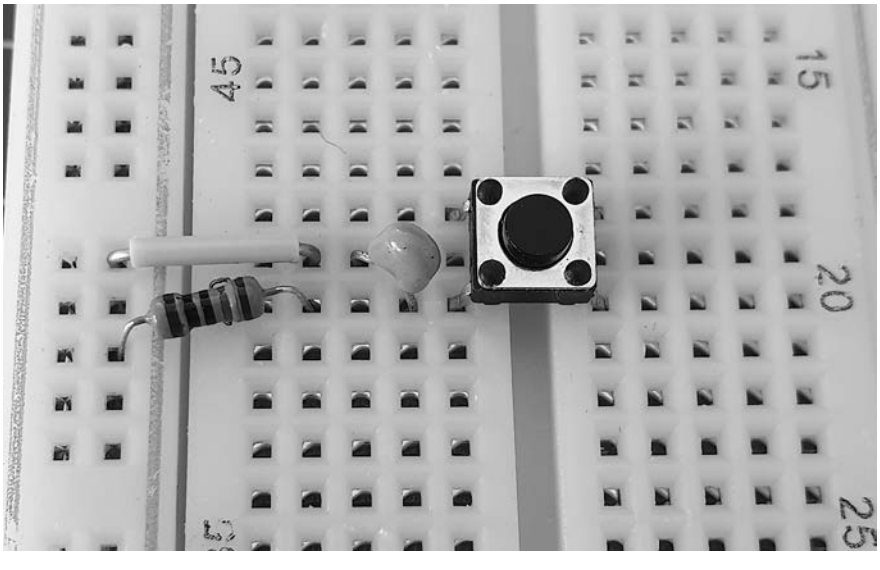


Рис. 4.21. Кнопка, конденсатор и резистор номиналом 10 кОм

3. Соедините отрезком провода контакт 5 V на плате Arduino с горизонтальным рядом на макетной плате, в который вставлен правый верхний контакт кнопки. Соедините другим отрезком провода контакт GND на плате Arduino с вертикальным рядом на макетной плате, к которому левыми концами подключены резистор и отрезок провода. Результат показан на рис. 4.22.

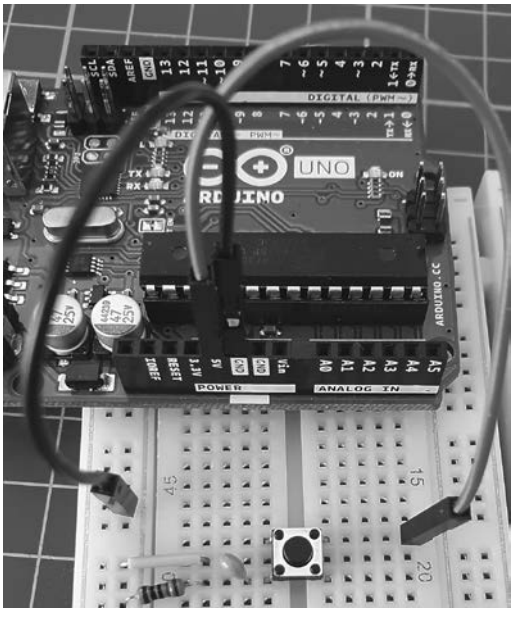


Рис. 4.22. Провода питания: 5 В (красный) и «земля» (черный)

4. Соедините проводом цифровой вывод 7 на Arduino с горизонтальным рядом на макетной плате, в который вставлен правый нижний контакт кнопки (рис. 4.23).

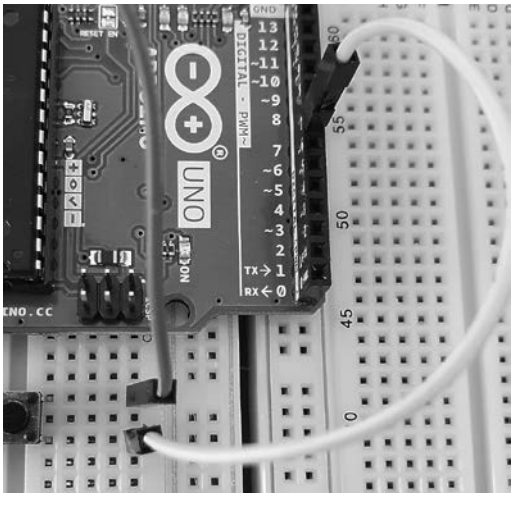


Рис. 4.23. Соединение кнопки с цифровым входом

5. Вставьте светодиод в макетную плату коротким выводом (катодом) в один из разъемов вертикального ряда GND, а длинным выводом (анодом) — в разъем горизонтального, правее. После подсоедините резистор номиналом 560 Ом — в разъем справа от светодиода, как показано на рис. 4.24.

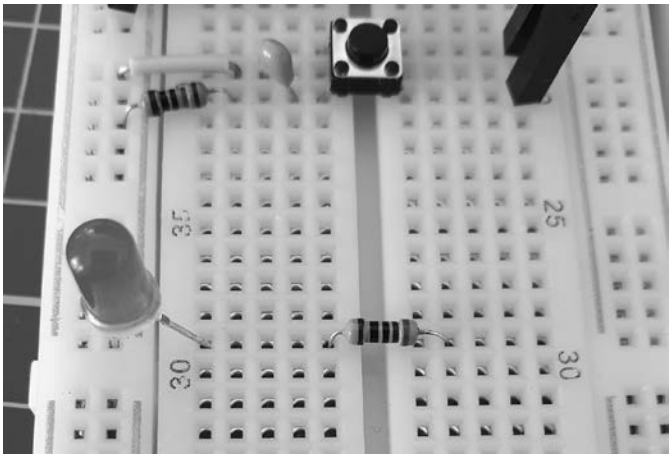


Рис. 4.24. Включение светодиода и резистора номиналом 560 Ом

6. Соедините отрезком провода правый конец резистора 560 Ом и цифровой вывод 3 на плате Arduino, как показано на рис. 4.25.

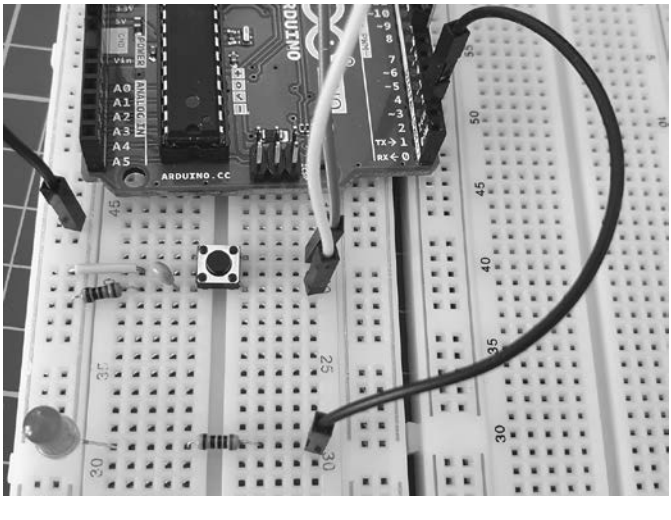


Рис. 4.25. Соединение светодиода с платой Arduino

Прежде чем продолжить, проверьте еще раз собранную цепь и убедитесь, что все компоненты соединены правильно. Сравните принципиальную схему с собранной цепью.

Скетч

Введите скетч из листинга 4.1 и загрузите его в плату.

Листинг 4.1. Цифровой вход

```
// Проект 4 — демонстрация работы цифрового входа
❶ #define LED 3
  #define BUTTON 7
  void setup()
  {
❷   pinMode(LED, OUTPUT);   // Выход для управления светодиодом
    pinMode(BUTTON, INPUT); // Вход для кнопки
  }

  void loop()
  {
    if ( digitalRead(BUTTON) == HIGH )
    {
      digitalWrite(LED, HIGH); // Включить светодиод
      delay(500);              // Ждать 0,5 секунды
      digitalWrite(LED, LOW);  // Выключить светодиод
    }
  }
```

Загрузив скетч, нажмите и сразу отпустите кнопку (светодиод должен зажечься на полсекунды).

Анализ скетча

Изучим новые элементы, появившиеся в скетче для проекта 4: инструкцию `#define`, использование цифрового ввода и оператор `if`.

Определение констант с помощью #define

Перед функцией `void setup()` находятся инструкции `#define` ❶. Они определяют фиксированные значения: во время компиляции скетча IDE заменит все вхождения определяемых имен их числовыми значениями. Например, повстречав слово `LED` в строке ❷, она заменит его на `3`. Заметьте, что мы не используем точку с запятой после числа в операторе `#define`.

Команду `#define` мы будем использовать в основном для обозначения в скетчах цифровых входов/выходов, к которым подключены светодиоды. Вообще, номера контактов и другие фиксированные значения (такие как время задержки) в скетчах предпочтительнее определять именно так. Это удобно, ведь, если значение используется в скетче несколько раз и позднее потребуется изменить его, вам не придется править его в разных местах. В этом примере константа `LED` используется трижды. Чтобы исправить ее значение, достаточно лишь изменить определение в инструкции `#define`.

Чтение состояний цифровых входов

Чтобы получить возможность читать состояние кнопки, в функции `void setup()` контакт сначала настраивается на работу в режиме цифрового входа:

```
pinMode(BUTTON, INPUT); // Вход для кнопки
```

Дальше, чтобы определить, соединяет ли кнопка цифровой вход с источником напряжения 5 В (нажата ли кнопка), вызывается функция `digitalRead(pin)`, где *`pin`* — номер цифрового входа, состояние которого нужно прочитать. Функция возвращает значение `HIGH` (напряжение на контакте близко к 5 В) или `LOW` (напряжение на контакте близко к 0 В).

Принятие решений с помощью if

Оператор `if` позволяет принимать решения в скетчах и, в зависимости от принятого решения, выполнять разный код. В скетче для проекта 4 использован код из листинга 4.2.

Листинг 4.2. Простой пример использования оператора if-then

```
if (digitalRead(BUTTON) == HIGH)
{
  digitalWrite(LED, HIGH); // Включить светодиод
  delay(500);              // Ждать 0,5 секунды
  digitalWrite(LED, LOW);  // Выключить светодиод
}
```

Первая строка в листинге 4.2 начинается с проверки условия в операторе `if`. Если оно истинно (то есть напряжение имеет величину `HIGH`), то кнопка нажата и можно выполнить код в фигурных скобках.

Для определения факта нажатия кнопки (когда `digitalRead(BUTTON)` возвращает значение `HIGH`) мы использовали *оператор сравнения* (`==`). Если заменить `==`

оператором != (не равно), светодиод будет выключаться в момент нажатия кнопки. Измените скетч, чтобы убедиться, что это действительно так.

ПРИМЕЧАНИЕ

Использование для сравнения единственного знака равенства (=), который означает «сделать равным», вместо двух (==), означающих «проверить равенство», — типичная ошибка. Вы можете не получить сообщения об ошибке, но оператор if будет работать неправильно!

Добившись успеха, попробуйте изменить в скетче продолжительность интервала, в течение которого светодиод остается включенным, или добавьте управление включением светодиода кнопкой в проект 3 (см. главу 3) (не разбирайте пока эту схему; она нам еще понадобится в следующем примере).

Доработка скетча: принятие альтернативных решений с помощью if-then-else

В оператор `if`, в ветвь `else`, можно добавить альтернативную последовательность действий. К примеру, если переписать листинг 4.1, добавив `else`, как показано в листинге 4.3, то светодиод будет включаться, *если* кнопка нажата. Иначе он будет выключаться. Использование `else` вынуждает Arduino выполнить второй фрагмент кода, если проверка в операторе `if` вернула ложное значение.

Листинг 4.3. Дополнительная ветвь else

```
#define LED 3
#define BUTTON 7

void setup()
{
  pinMode(LED, OUTPUT);   // Выход для управления светодиодом
  pinMode(BUTTON, INPUT); // Вход для кнопки
}

void loop()
{
  if ( digitalRead(BUTTON) == HIGH )
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
  }
}
```

Логические переменные

Иногда нужно запомнить некоторую характеристику, имеющую только два состояния (включено/выключено или горячо/холодно). *Логическая (булева) переменная* — легендарный компьютерный «бит», который может принимать только два значения: ноль (0, ложь) или один (1, истина). Подобно любым другим переменным, они должны объявляться перед использованием:

```
boolean raining = true; // Создать переменную raining (дождливо)
                        // и присвоить ей начальное значение true
```

Изменить состояние логической переменной в скетче можно простым присваиванием:

```
raining = false;
```

Поскольку логические переменные могут принимать только два значения (истина или ложь), их удобно использовать для принятия решений в операторе `if`. Логические переменные можно сравнивать со значениями `true` и `false` с помощью операторов `!=` и `==`:

```
if ( raining == true )
{
  if ( summer != true )
  {
    // Идет дождь, и сейчас не лето
  }
}
```

Операторы сравнения

Для принятия решений на основе значений двух или более логических переменных или других состояний используется несколько дополнительных операторов. В их число входят операторы НЕ (`!`), И (`&&`) и ИЛИ (`||`).

Оператор НЕ

Оператор НЕ обозначается восклицательным знаком (`!`). Он используется как сокращенная форма проверки того, что некоторое состояние НЕ истинно:

```
if ( !raining )
{
  // Дождя нет (raining == false)
}
```

Оператор И

Логический оператор И обозначается как `&&`. Он помогает уменьшить число отдельных проверок с помощью оператора `if`:

```
if (( raining == true ) && ( !summer ))
{
  // Идет дождь, и сейчас не лето (raining == true И summer == false)
}
```

Оператор ИЛИ

Логический оператор ИЛИ обозначается как `||`. Использовать его очень просто:

```
if (( raining == true ) || ( summer == true ))
{
  // Этот код выполняется в дождь или летом
}
```

Выполнение двух и более сравнений

В одном операторе `if` можно выполнить два или более сравнения:

```
if ( snow == true && rain == true && !hot )
{
  // Этот код выполняется, когда идет снег с дождем и не жарко
}
```

Для определения порядка выполнения операций используются круглые скобки. В следующем примере оператор `if` сначала проверит истинность или ложность условия в круглых скобках, а потом выполнит последнее сравнение:

```
if (( snow == true || rain == true ) && hot == false))
{
  // Этот код выполняется, когда идет снег или дождь и не жарко
}
```

Как и в примерах с оператором НЕ (`!`), простые проверки на истинность или ложность могут выполняться без сравнения с помощью `== true` или `== false`. Следующий код действует точно так же, как в предыдущем примере:

```
if (( snow || rain ) && !hot )
{
  // Этот код выполняется, когда идет снег или дождь и не жарко
  // ("идет снег" – истинно ИЛИ "идет дождь" – истинно) И НЕ жарко
}
```

Как видите, с помощью операторов сравнения и логических переменных можно организовать принятие решений, проверяя самые разные условия. Перейдя к более сложным проектам, вы по достоинству оцените эту возможность.

Проект 5: управление движением

Теперь применим вновь приобретенные знания. Представьте, что вы градостроитель и у вас есть мост через реку с единственной полосой для движения автотранспорта. Каждую неделю по ночам происходит одно-два ДТП, когда уставшие водители мчатся через мост, забывая убедиться в том, что путь свободен. Вы предложили установить светофор, но чиновники хотят увидеть на макете, как он будет действовать, прежде чем выделять средства. Можно было бы арендовать переносные светофоры, но они дорогие. Поэтому вы решили сконструировать модель моста с действующим светофором, собранным на светодиодах и плате Arduino.

Цель

Наша цель — установить трехцветные светофоры на обоих концах моста. Они должны разрешать движение только в одном направлении в каждый конкретный момент времени. Когда датчики на одном конце моста обнаруживают автомобиль, ожидающий включения зеленого, светофоры должны переключиться и разрешить движение.

Алгоритм

Для имитации датчиков обнаружения автомобилей на обоих концах моста мы используем две кнопки. Каждый светофор будет состоять из красных, желтых и зеленых светодиодов. Изначально система разрешает движение с запада на восток. Поэтому на светофоре, обращенном на запад, должен гореть зеленый свет, а на противоположном — красный.

Когда к мосту приближается автомобиль (моделируется нажатием кнопки) и на светофоре горит красный, система должна переключить свет на другом конце с зеленого на желтый, а потом на красный. После этого она должна выждать некоторое время, чтобы позволить автомобилям на мосту завершить его пересечение. Теперь на стороне с ожидающим автомобилем должен включиться желтый мигающий свет, чтобы потом смениться зеленым. Зеленый свет должен гореть, пока на противоположной стороне не появится автомобиль. После процесс должен повториться в обратном направлении.

Оборудование

Ниже перечислено оборудование для реализации этого проекта:

- два красных светодиода (LED1 и LED2);
- два желтых светодиода (LED3 и LED4);

- два зеленых светодиода (LED5 и LED6);
- шесть резисторов номиналом 560 Ом (R1–R6);
- два резистора номиналом 10 кОм (R7 и R8);
- два конденсатора емкостью 100 нФ (C1 и C2);
- две кнопки без фиксации (S1 и S2);
- одна макетная плата среднего размера;
- одна плата Arduino и кабель USB;
- несколько отрезков провода разной длины.

Схема

Поскольку предполагается управлять всего шестью светодиодами и принимать сигналы с двух кнопок, схема проекта достаточно проста (рис. 4.26).

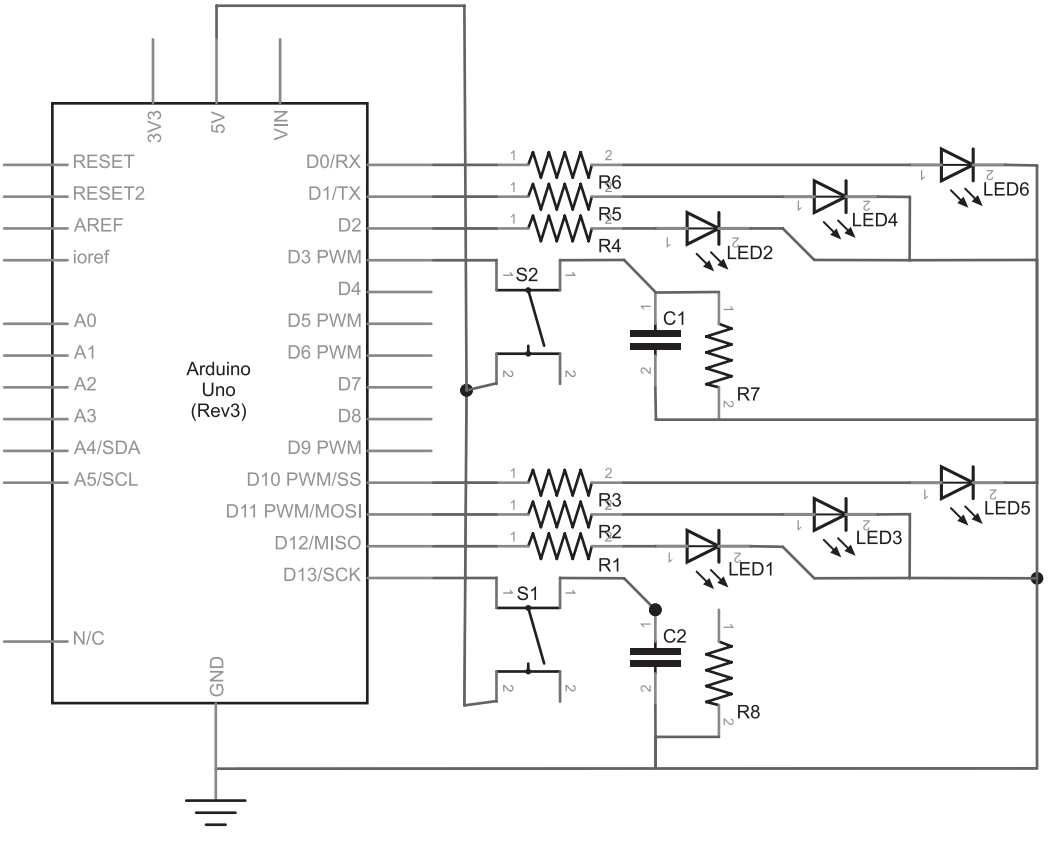


Рис. 4.26. Принципиальная схема для проекта 5

Эта схема сложнее версии с кнопкой и светодиодом из проекта 4 и имеет больше резисторов, больше светодиодов и еще одну кнопку.

Будьте внимательны при установке светодиодов. Постарайтесь не перепутать полярность: резисторы должны подключаться к анодам светодиодов, а катоды светодиодов — к контакту GND на плате Arduino, как показано на рис. 4.27.

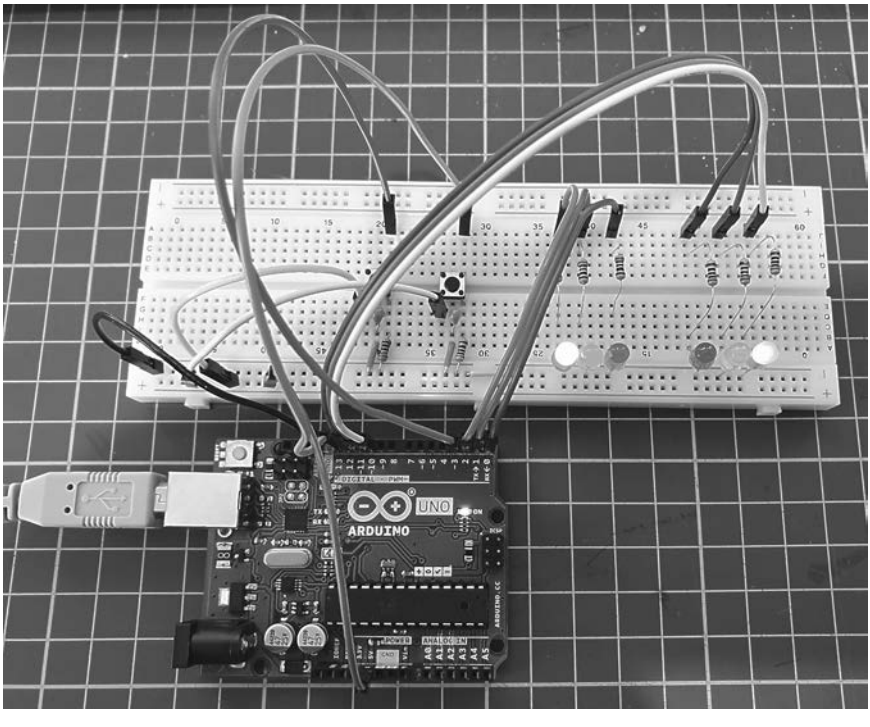


Рис. 4.27. Собранная схема

Скетч

А теперь перейдем к скетчу. Сможете ли вы самостоятельно проверить его соответствие алгоритму?

```
// Проект 5 — управление движением

// Определение контактов для подключения кнопок и светодиодов:
❶ #define westButton 3
#define eastButton  13
#define westRed     2
#define westYellow  1
```

```
#define westGreen   0
#define eastRed     12
#define eastYellow  11
#define eastGreen   10

#define yellowBlinkTime 500 // Период мигания желтого света 0,5 секунды

❷ boolean trafficWest = true; // запад = true, восток = false
❸ int flowTime = 10000;       // Период ожидания, чтобы пропустить
                              // автомобили, уже находящиеся на мосту
❹ int changeDelay = 2000;     // Задержка перед сменой цвета

void setup()
{
  // Настройка цифровых входов/выходов
  pinMode(westButton, INPUT);
  pinMode(eastButton, INPUT);
  pinMode(westRed, OUTPUT);
  pinMode(westYellow, OUTPUT);
  pinMode(westGreen, OUTPUT);
  pinMode(eastRed, OUTPUT);
  pinMode(eastYellow, OUTPUT);
  pinMode(eastGreen, OUTPUT);

  // Начальное состояние светофоров — зеленый на западной стороне
  digitalWrite(westRed, LOW);
  digitalWrite(westYellow, LOW);
  digitalWrite(westGreen, HIGH);
  digitalWrite(eastRed, HIGH);
  digitalWrite(eastYellow, LOW);
  digitalWrite(eastGreen, LOW);
}

void loop()
{
  // Запрошено движение с запада на восток?
  if ( digitalRead(westButton) == HIGH )
  {
    // Продолжать, только если движение меняется
    // на противоположное
    if ( trafficWest != true )
    {
      trafficWest = true; // Изменить флаг направления запад -> восток
      delay(flowTime);    // Дать автомобилям время пересечь мост
      digitalWrite(eastGreen, LOW);   // На восточной стороне погасить
      digitalWrite(eastYellow, HIGH); // зеленый сигнал, зажечь желтый
      delay(changeDelay);             // и затем красный
      digitalWrite(eastYellow, LOW);
      digitalWrite(eastRed, HIGH);
      delay(changeDelay);
      for ( int a = 0; a < 5; a++ )   // Воспроизвести мигающий желтый
```

```
      {
        digitalWrite(westYellow, LOW);
        delay(yellowBlinkTime);
        digitalWrite(westYellow, HIGH);
        delay(yellowBlinkTime);
      }
      digitalWrite(westYellow, LOW);
      digitalWrite(westRed, LOW);    // Сменить сигнал на западной
      digitalWrite(westGreen, HIGH); // стороне с красного на зеленый
    }
  }

  // Запрошено движение с востока на запад?
  if ( digitalRead(eastButton) == HIGH )
  {
    // Продолжать, только если движение меняется
    // на противоположное
    if ( trafficWest == true )
    {
      trafficWest = false; // Изменить флаг направления восток -> запад
      delay(flowTime);     // Дать автомобилям время пересечь мост
      digitalWrite(westGreen, LOW);

      // На восточной стороне сменить зеленый
      // сигнал на желтый и затем на красный
      digitalWrite(westYellow, HIGH);
      delay(changeDelay);
      digitalWrite(westYellow, LOW);
      digitalWrite(westRed, HIGH);
      delay(changeDelay);
      for ( int a = 0 ; a < 5 ; a++ ) // Воспроизвести мигающий желтый
      {
        digitalWrite(eastYellow, LOW);
        delay(yellowBlinkTime);
        digitalWrite(eastYellow, HIGH);
        delay(yellowBlinkTime);
      }
      digitalWrite(eastYellow, LOW);
      digitalWrite(eastRed, LOW);    // Сменить сигнал на восточной
      digitalWrite(eastGreen, HIGH); // стороне с красного на зеленый
    }
  }
}
```

Скетч начинается с определения ❶ соответствий между номерами контактов цифровых входов/выходов и смысловыми именами светодиодов и двух кнопок. С каждой стороны моста у нас есть красный, желтый, зеленый светодиоды и кнопка. Логическая переменная `trafficWest` ❷ хранит текущее направление движения по мосту — `true` (с запада на восток) и `false` (с востока на запад).

ПРИМЕЧАНИЕ

Обратите внимание, что trafficWest — единственная логическая переменная, определяющая направление движения как true или false. Наличие единственной переменной, как здесь, вместо двух (одна для направления на восток, другая — на запад) гарантирует невозможность одновременного выбора обоих направлений. Это поможет избежать аварий на мосту!

Целочисленная переменная `flowTime` ❸ определяет минимальный период времени для автомобилей, чтобы завершить движение по мосту. Когда со стороны, где горит красный свет, появляется машина, система выполняет задержку на это время, чтобы автомобили на мосту смогли закончить его пересечение. Целочисленная переменная `changeDelay` ❹ определяет период времени между переключениями светофора с зеленого на желтый и красный.

Прежде чем скетч вызовет функцию `void loop()`, в функции `void setup()` устанавливается направление движения с запада на восток.

Запуск скетча

После запуска скетч ничего не делает до нажатия одной из кнопок. Когда это произойдет на восточной стороне, следующая строка:

```
if ( trafficWest == true )
```

разрешит изменение сигналов светофора, только если движение по мосту осуществляется в противоположном направлении. Остальной код в этом разделе выполняет простую последовательность из задержек и команд включения/выключения разных светодиодов, имитирующую работу светофора.

Аналоговые и цифровые сигналы

В этом разделе вы узнаете, чем отличаются цифровые и аналоговые сигналы и как измерять величину последних на контактах аналоговых входов.

До сих пор мы использовали в своих скетчах только цифровые электрические сигналы с двумя дискретными уровнями. В частности, мы использовали `digitalWrite(`*`pin`*`, HIGH)` и `digitalWrite(`*`pin`*`, LOW)` для реализации мигания светодиода и `digitalRead()`, чтобы определить наличие (`HIGH`) или отсутствие (`LOW`) напряжения на контакте. На рис. 4.28 изображено визуальное представление цифрового сигнала, изменяющегося между высоким и низким уровнями.

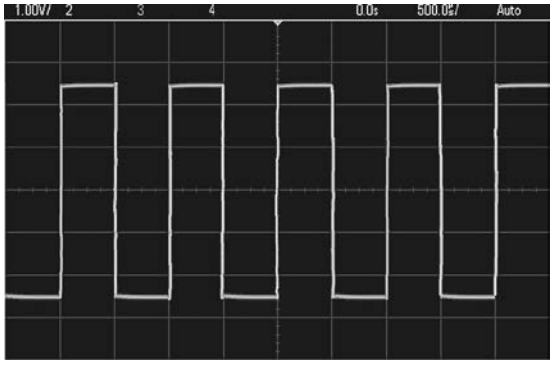


Рис. 4.28. Цифровой сигнал с уровнями HIGH, которые выглядят как горизонтальные отрезки в верхней части, и LOW — в нижней части

В отличие от цифровых аналоговые сигналы могут изменяться с неопределенным шагом между высоким и низким уровнем. На рис. 4.29 мы видим аналоговый сигнал в виде синусоиды. Заметьте, что со временем напряжение плавно меняется между высоким и низким уровнем.

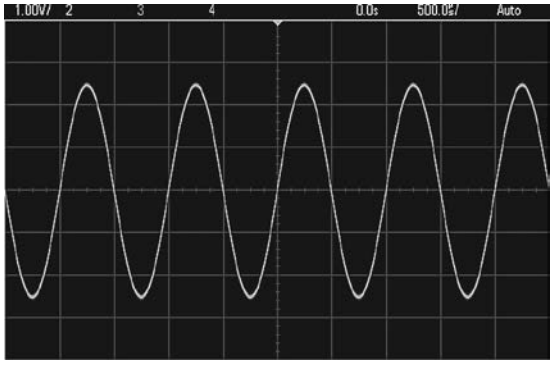


Рис. 4.29. Аналоговый сигнал в форме синусоиды

На плате Arduino высокий уровень близок к 5 В, низкий — к 0 В. У платы есть шесть аналоговых входов (рис. 4.30), с помощью которых можно измерять значения аналоговых сигналов. Эти входы позволяют безопасно измерять напряжение от 0 до 5 В.

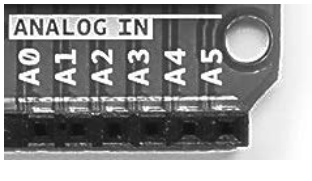


Рис. 4.30. Аналоговые входы на плате Arduino Uno

Если вызвать функцию `analogRead()`, она вернет число в диапазоне от 0 до 1023, пропорциональное напряжению, приложенному к контакту аналогового

входа. К примеру, с помощью `analogRead()` можно прочитать значение напряжения на аналоговом входе `A0` и сохранить его в целочисленной переменной `a`:

```
a = analogRead(0); // Прочитает напряжение на аналоговом входе 0 (A0)
                   // и вернет число в диапазоне от 0 до 1023, которое обычно
                   // соответствует диапазону от 0 до 4,995 вольта
```

Проект 6: тестер для одноэлементных батареек

В последнее время спрос на одноэлементные батарейки снижается. Тем не менее многие все еще пользуются бытовыми приборами и устройствами, питающимися от батареек типа AA, AAA, C или D (пульты дистанционного управления, часы или детские игрушки). Напряжение таких батареек намного меньше 5 В, поэтому его можно измерить с помощью Arduino для определения состояния элемента. В этом проекте мы создадим тестер для батареек.

Цель

Напряжение новых одноэлементных батареек, например, типа AA обычно около 1,6 В, и его уровень снижается по мере использования. В нашем проекте мы измерим напряжение на батарейке и отобразим степень ее пригодности при помощи светодиодов. Мы будем определять напряжение через аналоговый вход с помощью `analogRead()` и преобразовывать полученное значение в вольты. Максимальное напряжение, которое можно подать на аналоговый вход, равно 5 В. Поделив 5 на 1024 (количество возможных значений), мы получим число 0,0048, которое и будем использовать как коэффициент для преобразования прочитанного значения в вольты. То есть, если `analogRead()` вернет число 512, тогда, умножив его на 0,0048, мы получим напряжение 2,4576 В.

Алгоритм

Рассмотрим алгоритм работы тестера батареек.

1. Прочитать значение с аналогового входа.
2. Умножить прочитанное значение на коэффициент 0,0048 для получения величины напряжения в вольтах.
3. Если напряжение больше или равно 1,6 В, включить на короткий промежуток зеленый светодиод.
4. Если напряжение больше 1,4 В *и* меньше 1,6 В, включить на короткий промежуток желтый светодиод.
5. Если напряжение меньше 1,4 В, включить на короткий промежуток красный светодиод.
6. Повторять до бесконечности.

Оборудование

Ниже перечислено оборудование для реализации этого проекта:

- три резистора с номиналом 560 Ом (R1–R3);
- один зеленый светодиод (LED1);
- один желтый светодиод (LED2);
- один красный светодиод (LED3);
- одна макетная плата;
- несколько отрезков провода разной длины;
- одна плата Arduino и кабель USB.

Схема

На рис. 4.31 изображена принципиальная схема тестера для одноэлементных батареек. Обратите внимание на два контакта слева, подписанных как + и –. К этим контактам должны подключаться *с соблюдением полярности* проверяемые батарейки. Положительный полюс батарейки должен подключаться к положительному контакту (+), а отрицательный — к отрицательному (–).

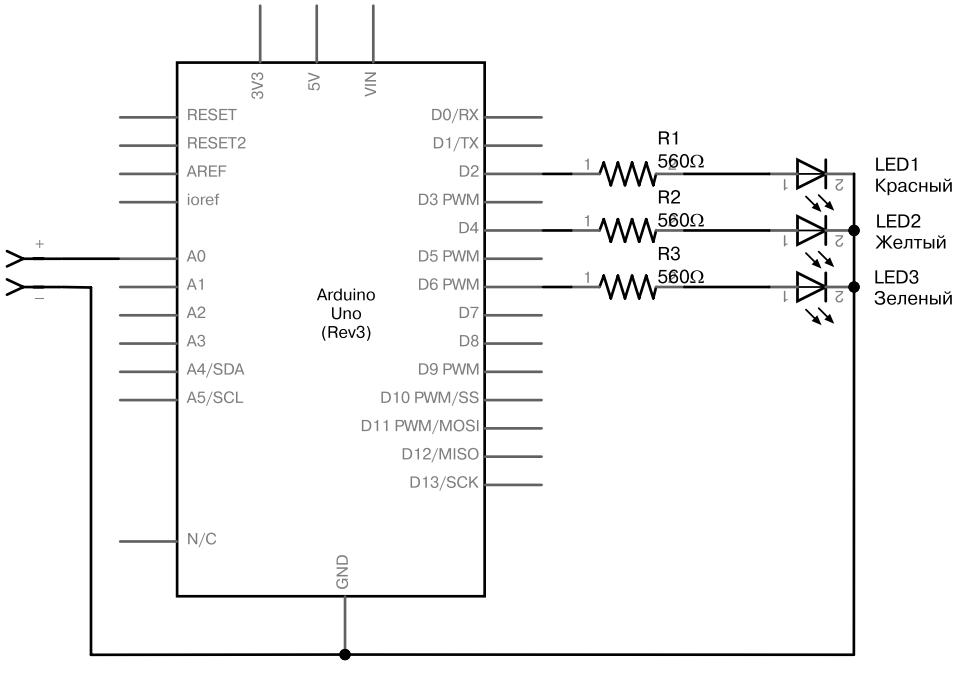


Рис. 4.31. Принципиальная схема для проекта 6

ВНИМАНИЕ

Ни при каких условиях не пытайтесь измерять напряжение выше 5 В и не подключайте положительный полюс батарейки к отрицательному контакту и наоборот, подобные действия выведут из строя вашу плату.

Скетч

А теперь скетч. Поскольку аналоговые значения могут быть дробными, в этом скетче мы используем новый тип переменных — *вещественные переменные*, или *переменные с плавающей точкой (float)*. Они могут хранить дробные значения:

```
// Проект 6 – тестер для одноэлементных батареек
#define newLED 2 // Зеленый светодиод 'новая'
#define okLED 4  // Желтый светодиод 'норма'
#define oldLED 6 // Красный светодиод 'старая'

int analogValue = 0;
❶ float voltage = 0;
int ledDelay = 2000;

void setup()
{
  pinMode(newLED, OUTPUT);
  pinMode(okLED, OUTPUT);
  pinMode(oldLED, OUTPUT);
}

void loop()
{
❷   analogValue = analogRead(0);
❸   voltage = 0.0048*analogValue;
❹   if ( voltage >= 1.6 )
    {
      digitalWrite(newLED, HIGH);
      delay(ledDelay);
      digitalWrite(newLED, LOW);
    }
❺   else if ( voltage < 1.6 && voltage > 1.4 )
    {
      digitalWrite(okLED, HIGH);
      delay(ledDelay);
      digitalWrite(okLED, LOW);
    }
❻   else if ( voltage <= 1.4 )
    {
      digitalWrite(oldLED, HIGH);
      delay(ledDelay);
      digitalWrite(oldLED, LOW);
    }
}
```

Скетч для проекта 6 читает значение аналогового входа 0 ❷ и преобразует его в напряжение ❸. С новым типом переменных — `float` ❶ — вы познакомитесь в следующем разделе, где обсуждаются арифметические операции и операторы сравнения чисел.

Выполнение арифметических операций в Arduino

Arduino, как и карманный калькулятор, выполняет разные арифметические операции: умножение, деление, сложение и вычитание. Ниже приводится несколько примеров:

```
a = 100;
b = a + 20;
c = b - 200;
d = c + 80; // d получит значение 0
```

Вещественные переменные

Когда нужно оперировать вещественными числами, в скетчах используются переменные типа `float`. В таких переменных можно хранить значения от $3{,}4028235 \times 10^{38}$ до $-3{,}4028235 \times 10^{38}$, точность которых ограничена 6–7 десятичными знаками. В вычислениях допускается смешивать целые и вещественные числа. Например, можно сложить вещественное число `f` с целым `a` и сохранить результат в вещественной переменной `g`:

```
int a = 100;
float f;
float g;

f = a / 3; // f = 33.333333
g = a + f; // g = 133.333333
```

Операторы сравнения чисел

Мы уже использовали операторы сравнения `==` и `!=` с инструкциями `if` в проекте 5, где определяли уровень входных цифровых сигналов. Есть еще несколько операторов, которые можно применять к числам или числовым переменным:

- `<` (меньше);
- `>` (больше);
- `<=` (меньше или равно);
- `>=` (больше или равно).

Мы использовали их в строках ❹, ❺ и ❻ в скетче для проекта 6, описанного выше.

Увеличение точности измерения аналоговых сигналов с помощью источника опорного напряжения

Как было показано в проекте 6, функция `analogRead()` возвращает значение, пропорциональное напряжению от 0 до 5 В. Верхнее значение 5 В (его называют *опорным напряжением*) — максимально допустимое напряжение, которое можно подавать на аналоговые входы Arduino и для которого возвращается высшее значение 1023.

Для увеличения точности измерения более низких напряжений задействуем низковольтный источник опорного напряжения. Например, когда опорное напряжение равно 5 В, функция `analogRead()` представляет измеренное напряжение как значение от 0 до 1023. Но если нужно измерять напряжение (например) не выше 2 В, можно заставить Arduino представлять числами 0–1023 диапазон от 0 до 2 В для повышения точности измерений. Нам необходимо задействовать внутренний или внешний источник опорного напряжения, как описывается далее.

Использование внешнего источника опорного напряжения

Первый метод — подача опорного напряжения на контакт AREF (*a*nalog *ref*erence — опорный аналоговый сигнал), как показано на рис. 4.32.

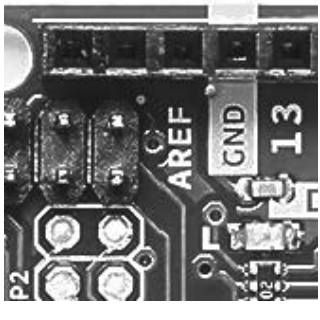


Рис. 4.32. Контакт AREF на плате Arduino Uno

Подать новое опорное напряжение можно, подключив положительный полюс внешнего источника питания к контакту AREF на плате Arduino, а отрицательный — к контакту GND. Помните, что опорным напряжением может быть только более низкое. Потому что опорное напряжение, подводимое к плате Arduino Uno, не должно превышать 5 В. Самый простой способ установить более низкое опорное напряжение — собрать делитель напряжения на двух резисторах (рис. 4.33).

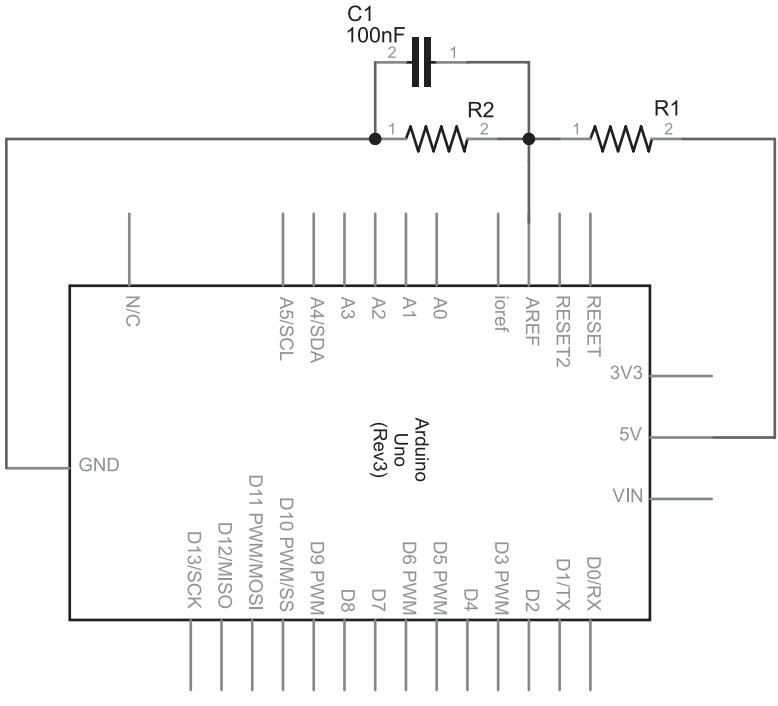


Рис. 4.33. Делитель напряжения

Величина опорного напряжения здесь определяется сопротивлением резисторов *R1* и *R2* по следующей формуле:

$$V_{out} = V_{in}\left(\frac{R2}{R1 + R2}\right),$$

где V_{out} — величина опорного напряжения, а V_{in} — входное напряжение (в данном случае 5 В); *R1* и *R2* — сопротивление резисторов в омах.

Простейший способ поделить напряжение — разбить V_{in} пополам, выбрав резисторы *R1* и *R2* с одинаковым сопротивлением — например, 10 кОм каждый. Для делителя лучше использовать резисторы с наименьшим допуском отклонений, например 1 %. Проверьте их истинное сопротивление мультиметром и используйте измеренные значения в вычислениях.

Еще можно добавить конденсатор емкостью 100 нФ между контактами AREF и GND, чтобы избавиться от шумов на контакте AREF и уменьшить нестабильность измерений.

При использовании внешнего источника опорного напряжения вставьте следующую строку в функцию `void setup()`:

```
analogReference(EXTERNAL); // Использовать контакт AREF в качестве
                           // источника опорного напряжения
```

Использование внутреннего источника опорного напряжения

В Arduino Uno есть и внутренний источник опорного напряжения 1,1 В. Если этот уровень вам подходит, то не придется использовать дополнительные электронные компоненты. Просто добавьте следующую строку в функцию `void setup()`:

```
analogReference(INTERNAL); // Выбрать внутренний источник
                           // опорного напряжения 1,1 В
```

Переменный резистор

Переменные резисторы *(потенциометры)* позволяют изменять их сопротивление от 0 Ом до соответствующего им номинального значения. На принципиальных схемах переменные резисторы обозначаются, как показано на рис. 4.34.

Рис. 4.34. Обозначение переменного резистора (потенциометра)

У таких резисторов есть три контакта: один центральный и по одному с каждого конца. По мере вращения оси переменного резистора сопротивление между одним крайним и центральным контактом увеличивается, а между другим крайним и центральным — уменьшается.

Переменные резисторы могут быть *линейными* или *логарифмическими*. Сопротивление переменного резистора с линейной характеристикой изменяется прямо пропорционально углу поворота, а с логарифмической характеристикой — сначала медленно, а потом все быстрее и быстрее. Логарифмические потенциометры чаще используются в звукоусилительной аппаратуре, так как они точнее соответствуют кривой восприятия человеческого уха. Отличить линейный потенциометр от логарифмического обычно можно по маркировке на тыльной стороне. У большинства из них рядом с номинальным значением сопротивления будет напечатан символ A или B: A — на логарифмических потенциометрах, B — на линейных. В большинстве проектов для Arduino используются линейные переменные резисторы (рис. 4.35).

Рис. 4.35. Типичный линейный переменный резистор

Бывают и миниатюрные переменные резисторы. Они часто называются *подстроечными* (рис. 4.36). Благодаря небольшим размерам такие резисторы удобно использовать в устройствах для регулировки и создания прототипов, потому что они отлично умещаются на макетной плате.

Рис. 4.36. Разные подстроечные резисторы

ПРИМЕЧАНИЕ

Приобретая подстроечные резисторы, обращайте внимание на их тип. Лучше всего использовать те, что легко регулируются отверткой. Кроме того, резисторы с закрытым корпусом, такие как на рис. 4.36, служат дольше своих более дешевых аналогов с открытыми контактами.

Пьезоэлектрические зуммеры

Пьезоэлектрический зуммер (или просто зуммер) — это маленькое устройство в цилиндрическом корпусе, которое можно использовать для подачи громких и раздражающих звуковых сигналов (вроде предупреждения об аварии или для забавы). На рис. 4.37 изображен зуммер TDK PS1240 рядом с 25-центовой монетой США[1], чтобы вы могли лучше представить его реальный размер.

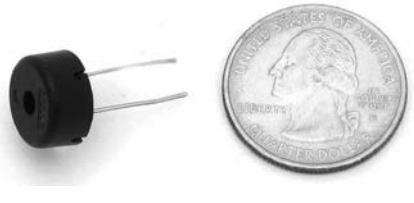


Рис. 4.37. Пьезоэлектрический зуммер TDK PS1240

[1] Диаметр 25-центовой монеты (24,3 мм) практически совпадает с диаметром современной пятирублевой (25 мм). — *Примеч. пер.*

Внутри зуммера есть очень тонкая пластина, которая изменяет форму при подаче напряжения. Когда ток подается на зуммер импульсами, она начинает вибрировать, генерируя звуковые волны определенной частоты.

Зуммер может пригодиться в наших проектах. Его можно быстро включать и выключать, как светодиод, имитируя импульсы. Пьезоэлементы не имеют полярности и могут подключаться как угодно.

Изображение пьезоэлектрических зуммеров на схемах

Значок, обозначающий зуммер на схеме, напоминает динамик (рис. 4.38), так проще его распознать.

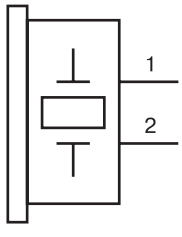


Рис. 4.38. Обозначение пьезоэлектрического зуммера на схемах

ПРИМЕЧАНИЕ

При покупке зуммера для нашего проекта убедитесь, что выбранная модель содержит только пьезоэлемент. Некоторые зуммеры похожи на изображенный на рис. 4.38, но имеют встроенную схему, генерирующую звук. Такой для этого проекта не подходит, потому что мы собираемся управлять высотой звука с помощью Arduino.

Проект 7: испытание пьезоэлектрического зуммера

Если у вас есть зуммер и желание опробовать его, то подключите его одним выводом к контакту GND, а другим — к контакту D3 на плате Arduino. Затем загрузите следующий демонстрационный скетч в свою плату:

```
// Проект 7 – испытание пьезоэлектрического зуммера
#define PIEZO 3 // Контакт 3 может выводить сигналы ШИМ
                // для управления высотой звука
int del = 500;

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

void loop()
{
❶   analogWrite(PIEZO, 128); // Сгенерировать импульсный сигнал ШИМ
                             // с коэффициентом заполнения 50 процентов
  delay(del);
  digitalWrite(PIEZO, LOW); // Выключить зуммер
  delay(del);
}
```

Этот скетч выводит сигнал с широтно-импульсной модуляцией на третий контакт. Изменяя коэффициент заполнения в вызове функции `analogWrite()`, который сейчас равен 128 (это составляет 50 %) ❶, можно управлять звучанием зуммера.

Для усиления громкости звучания увеличьте напряжение, приложенное к зуммеру. Сейчас максимальное напряжение ограничено 5 В, но зуммер будет издавать более громкий звук, если приложить к нему напряжение 9 или 12 В. Поскольку Arduino не может выдавать более высокое напряжение, вам понадобится внешний источник питания (например, 9-вольтовая батарея) и транзистор в качестве электронного ключа. При этом мы будем использовать тот же скетч и схему, изображенную на рис. 4.39.

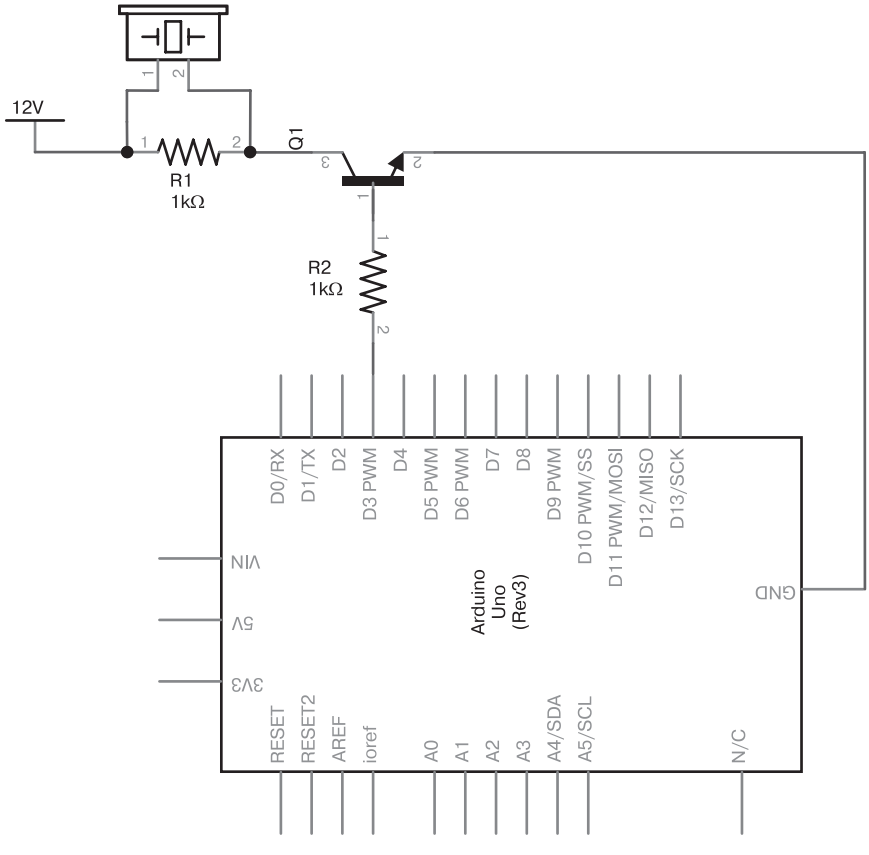


Рис. 4.39. Принципиальная схема для проекта *7*

Элемент на схеме, обозначенный как 12 V, — это точка подключения положительного полюса внешнего источника более высокого напряжения, отрицательный полюс которого должен быть подключен к контакту GND на плате Arduino.

Проект 8: быстродействующий термометр

Температура может быть представлена аналоговым сигналом, например генерируемым температурным датчиком TMP36 (рис. 4.40), который производится компанией Analog Devices (http://www.analog.com/tmp36/).

Рис. 4.40. Температурный датчик TMP36

Обратите внимание, что датчик TMP36 похож на транзистор BC548 для управления реле из главы 3. Этот датчик выдает напряжение, пропорциональное температуре, благодаря чему текущую температуру можно определять простым преобразованием. Например, 25 °C соответствует выходное напряжение 750 мВ, и каждое изменение температуры на градус влечет изменение напряжения на 10 мВ. Датчик TMP36 может измерять температуры в диапазоне от –40 до 125 °C.

Функция `analogRead()` возвращает значение от 0 до 1023, что соответствует диапазону напряжения от 0 до (почти) 5000 мВ (5 В). Если умножить результат вызова `analogRead()` на (5000/1024), получится напряжение, фактически возвращаемое датчиком. Далее нужно вычесть 500 (смещение, используемое датчиком TMP36 для обеспечения возможности измерения отрицательных температур) и разделить на 10. В результате получится температура в градусах Цельсия. Если вы предпочитаете шкалу Фаренгейта, умножьте температуру в Цельсиях на 1,8 и прибавьте 32.

Цель

В этом проекте мы используем датчик TMP36 для создания быстродействующего термометра. При температуре ниже +20 °C должен включиться синий светодиод. От +20 до +26 °C — зеленый, а когда температура превысит +26 °C, должен включаться красный светодиод.

Оборудование

Ниже перечислено оборудование для этого проекта:

- три резистора номиналом 560 Ом (R1–R3);
- один красный светодиод (LED1);
- один зеленый светодиод (LED2);

- один синий светодиод (LED3);
- один датчик температуры TMP36;
- одна макетная плата;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Схема

Схема устройства очень проста. Если смотреть на датчик температуры TMP36 со стороны маркировки, вывод слева должен подключаться к источнику напряжения 5 В, центральный вывод — это выход напряжения, соответствующего температуре, и вывод справа должен подключаться к «земле» (GND), как показано на рис. 4.41.

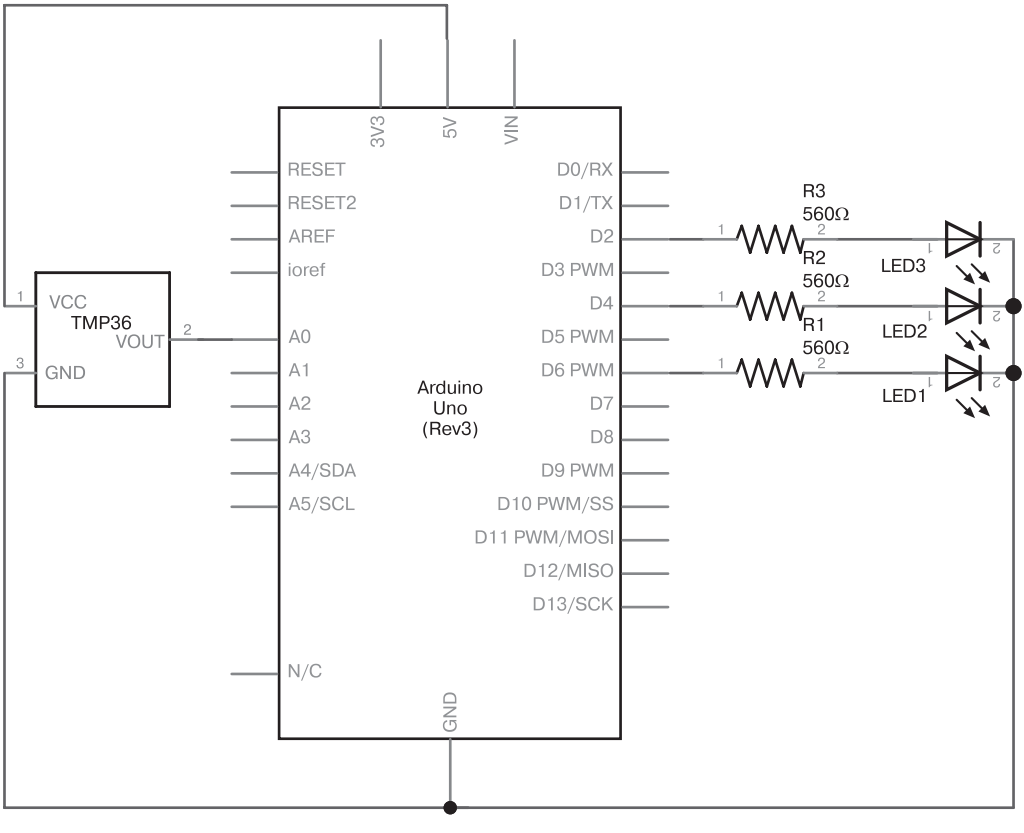


Рис. 4.41. Принципиальная схема для проекта 8

Скетч

А теперь скетч:

```
// Проект 8 — быстродействующий термометр

// Определение контактов подключения светодиодов:
#define HOT    6
#define NORMAL 4
#define COLD   2

float voltage = 0;
float celsius = 0;
float hotTemp  = 26;
float coldTemp = 20;
float sensor = 0;

void setup()
{
  pinMode(HOT, OUTPUT);
  pinMode(NORMAL, OUTPUT);
  pinMode(COLD, OUTPUT);
}

void loop()
{
  // Считать напряжение с датчика и преобразовать в градусы Цельсия
❶ sensor = analogRead(0);
  voltage = (sensor*5000)/1024; // Преобразовать в милливольты
  voltage = voltage-500;        // Учесть смещение
  celsius = voltage/10;         // Преобразовать милливольты в градусы

  // Выполнить действия, соответствующие измеренной температуре
❷ if ( celsius < coldTemp )
  {
    digitalWrite(COLD, HIGH);
    delay(1000);
    digitalWrite(COLD, LOW);
  }
❸ else if ( celsius > coldTemp && celsius <= hotTemp )
  {
    digitalWrite(NORMAL, HIGH);
    delay(1000);
    digitalWrite(NORMAL, LOW);
  }
  else
  {
    // celsius > hotTemp
    digitalWrite(HOT, HIGH);
    delay(1000);
    digitalWrite(HOT, LOW);
  }
}
```

Сначала скетч считывает напряжение с датчика и преобразует его в градусы Цельсия ❶. После с помощью оператора `if-else` (❷ и ❸) текущая температура сравнивается с границами холодного и жаркого диапазонов, и включается соответствующий светодиод. Инструкции `delay(1000)` предотвращают слишком быстрое переключение светодиодов, когда температура колеблется на границе двух диапазонов.

Поэкспериментируйте с термометром, обдувая его прохладным воздухом для понижения температуры или зажав корпус TMP36 между пальцами для ее повышения.

Что дальше?

Вот и конец четвертой главы. Теперь ваш арсенал пополнился такими дополнительными инструментами, как цифровые входы и выходы, новые типы переменных и разнообразные математические функции. Дальше вы продолжите развлекаться со светодиодами, научитесь создавать свои функции, сконструируете компьютерную игру с электронным кубиком и многое другое.

5

Функции

В этой главе вы:

- научитесь создавать свои функции;
- узнаете, как принимать решения с помощью циклов `while` и `do-while`;
- освоите обмен данными между платой Arduino и монитором последовательного порта;
- познакомитесь с переменными типа `long`.

Здесь вы узнаете о новых методах, облегчающих чтение скетчей для Arduino и упрощающих их проектирование за счет создания своих функций. Вы научитесь писать модульный код, который сможете использовать в следующих проектах. Кроме того, в этой главе вы узнаете, как принимать решения, управляющие выполнением блоков кода, и встретитесь с новым типом целочисленных переменных `long`. И в итоге попробуете использовать собственные функции для реализации термометра нового типа.

Функция состоит из последовательности инструкций, упакованных вместе в один именованный модуль, который можно вызвать из любой точки в скетче. Несмотря на наличие множества встроенных функций в языке Arduino, не всегда удается найти ту, что отвечает конкретным потребностям. В некоторых случаях возникает необходимость многократно использовать одну последовательность инструкций в разных частях скетча. Это влечет за собой неэффективное расходование памяти. В обоих случаях хочется иметь функцию, выполняющую необходимые действия. Должен сообщить, что ее искать не нужно — вы можете написать эту функцию самостоятельно.

Проект 9: программирование функции для выполнения повторяющихся действий

Итак, рассмотрим простую функцию, повторно выполняющую некоторые действия по мере необходимости. К примеру, следующая функция дважды включает (❶ и ❸) и выключает (❷ и ❹) встроенный светодиод:

```
  void blinkLED()
  {
❶   digitalWrite(13, HIGH);
    delay(1000);
❷   digitalWrite(13, LOW);
    delay(1000);
❸   digitalWrite(13, HIGH);
    delay(1000);
❹   digitalWrite(13, LOW);
    delay(1000);
}
```

А теперь используем ее в законченном скетче, который можно загрузить в Arduino:

```
// Проект 9 — создание функции для повторного выполнения действий

#define LED 13
#define del 200

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

void blinkLED()
{
  digitalWrite(LED, HIGH);
  delay(del);
  digitalWrite(LED, LOW);
  delay(del);
  digitalWrite(LED, HIGH);
  delay(del);
  digitalWrite(LED, LOW);
  delay(del);
}

void loop()
{
❶   blinkLED();
    delay(1000);
}
```

Когда внутри функции `void loop()` вызывается `blinkLED()` ❶, Arduino выполняет команды внутри `void blinkLED()`. Другими словами, вы создали свою функцию и использовали ее по мере необходимости.

Проект 10: функция, изменяющая число миганий светодиода

Созданная нами функция довольно ограниченна. Представьте, что вам понадобилось изменить число миганий или задержку. Это легко сделать — напишем функцию для изменения значений:

```
void blinkLED(int cycles, int del)
{
  for ( int z = 0 ; z < cycles ; z++ )
  {
    digitalWrite(LED, HIGH);
    delay(del);
    digitalWrite(LED, LOW);
    delay(del);
  }
}
```

Наша новая функция `void blinkLED()` принимает два целочисленных значения: `cycles` (определяет, сколько раз должен мигнуть светодиод) и `del` (время задержки между включением и выключением светодиода). Если понадобится мигнуть светодиодом 12 раз с задержкой 100 миллисекунд, вы выполните вызов `blinkLED(12, 100)`. Введите следующий скетч в IDE и поэкспериментируйте с функцией:

```
// Проект 10 — функция, изменяющая число миганий

#define LED 13

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

void blinkLED(int cycles, int del)
{
  for ( int z = 0 ; z < cycles ; z++ )
  {
    digitalWrite(LED, HIGH);
    delay(del);
    digitalWrite(LED, LOW);
    delay(del);
  }
}
```

```
void loop()
{
❶  blinkLED(12, 100);
  delay(1000);
}
```

В строке ❶ можно видеть, что нашей функции `blinkLED()` передаются значения `12` и `100` (число миганий и величина задержки). Как результат, светодиод мигнет 12 раз через каждые 100 миллисекунд. Потом произойдет задержка на 1000 миллисекунд (1 секунду), после чего функция `loop()` начнет все сначала.

Функция, возвращающая значения

Мы можем создавать функции, не только принимающие значения в виде параметров (как `void blinkLED()` в проекте 10), но и возвращающие их. Так делает функция `analogRead()`, возвращающая число от 0 до 1023, пропорциональное напряжению на аналоговом входе. Ее мы видели в проекте 8.

Все функции до этих пор начинались с ключевого слова `void`. Оно означает, что функция ничего не возвращает, то есть возвращает значение `void` (ничего). Но мы можем создавать функции, возвращающие значения любого нужного нам типа. Например, если понадобится, чтобы функция возвращала целое число, мы должны начать ее объявление с ключевого слова `int`. Если нужна функция, возвращающая вещественное значение, мы начнем ее объявление со слова `float`. Создадим несколько полезных функций, возвращающих фактические значения.

Взгляните на следующую функцию, которая преобразует градусы Цельсия в градусы Фаренгейта:

```
float convertTemp(float celsius)
{
  float fahrenheit = 0;
  fahrenheit = (1.8 * celsius) + 32;
  return fahrenheit;
}
```

В первой строке определяется имя функции (`convertTemp`), тип возвращаемого ею значения (`float`) и описание всех принимаемых параметров (`float celsius`). Чтобы воспользоваться этой функцией, ей нужно передать некоторое значение. К примеру, если нужно преобразовать 40 градусов Цельсия в градусы Фаренгейта и сохранить результат в переменной типа `float` с именем `tempf`, это можно сделать так:

```
tempf = convertTemp(40);
```

Эта инструкция поместит число `40` в переменную `celsius` внутри `convertTemp`, вызовет функцию для выполнения вычислений `fahrenheit = (1.8 * celsius) + 32` и сохранит в переменной `tempf` результат, возвращаемый из `convertTemp` строкой `return fahrenheit`.

Проект 11: быстродействующий термометр, сообщающий температуру миганием светодиода

Теперь применим знания по написанию собственных функций и реализуем следующий проект — быстродействующий термометр на основе датчика температуры TMP36 из главы 4 и светодиода на плате Arduino. Если температура ниже +20 °C, светодиод должен мигнуть дважды и сделать паузу. Если температура будет от +20 до +26 °C, светодиод должен мигнуть четыре раза и сделать паузу. Если температура выше +26 °C, светодиод должен мигнуть шесть раз.

Мы сделаем наш скетч более модульным, разбив его на отдельные функции. Благодаря этому его проще будет понять, а функции будут использоваться многократно. Наш термометр решает две основные задачи: измеряет и классифицирует температуру и мигает светодиодом определенное число раз (в зависимости от температуры).

Оборудование

Нам понадобится минимальный набор оборудования:

- один температурный датчик TMP36;
- одна макетная плата;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Схема

Схема очень проста (рис. 5.1).

Скетч

Для скетча нужно написать две функции. Первая будет читать значение с датчика TMP36, преобразовывать его в градусы Цельсия и возвращать число 2, 4 или 6, соответствующее числу миганий светодиода. Для решения этой задачи примем за основу скетч из проекта 8.

Чтобы решить вторую задачу, возьмем функцию `blinkLed()` из проекта 9. `void loop` будет вызывать упомянутые функции и выполнять паузу в 2 секунды перед перезапуском.

ПРИМЕЧАНИЕ

Не забудьте сохранить измененные скетчи проектов в файлах с другими именами, чтобы случайно не потерять плоды своих трудов!

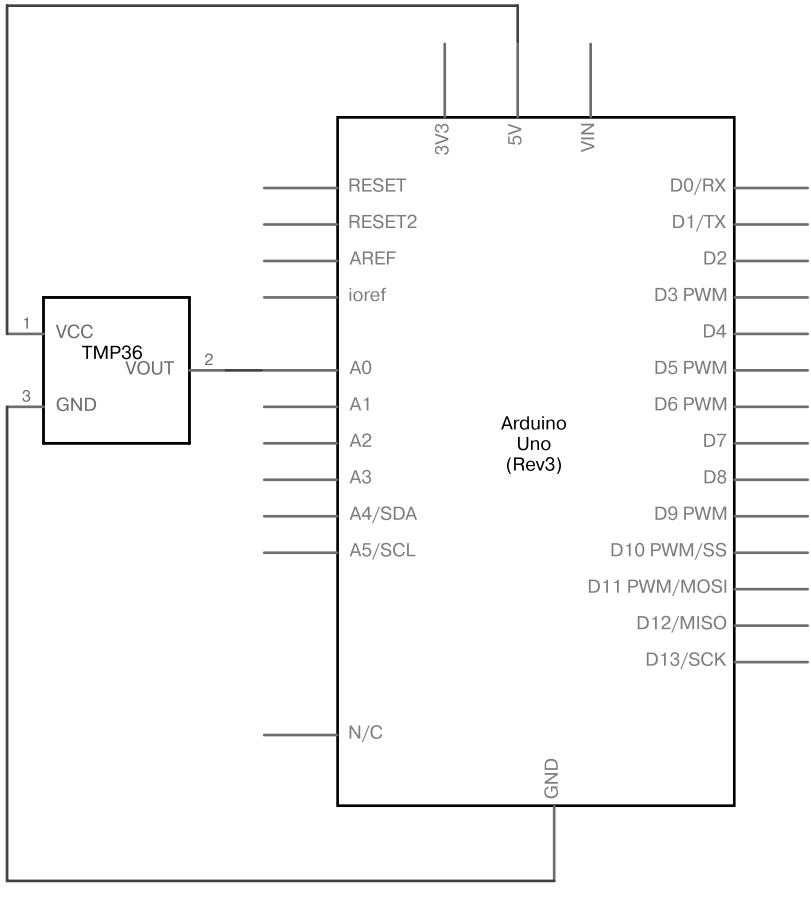


Рис. 5.1. Принципиальная электрическая схема для проекта 11

Введите следующий скетч в IDE:

```
// Проект 11 — быстродействующий термометр,
//             сообщающий температуру миганием

#define LED 13

int blinks = 0;

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

int checkTemp()
```

```
{
  float voltage = 0;
  float celsius = 0;
  float hotTemp = 26;
  float coldTemp = 20;
  float sensor = 0;
  int result;

  // Прочитать значение с датчика и преобразовать в градусы Цельсия
  sensor = analogRead(0);
  voltage = (sensor*5000)/1024; // Преобразовать в милливольты
  voltage = voltage-500;        // Учесть смещение
  celsius = voltage/10;         // Преобразовать милливольты в градусы

  // Выполнить действия для разных диапазонов температур
  if (celsius < coldTemp)
  {
    result = 2;
  }
  else if (celsius >= coldTemp && celsius <= hotTemp)
  {
    result = 4;
  }
  else
  {
    result = 6; // (celsius > hotTemp)
  }
  return result;
}

void blinkLED(int cycles, int del)
{
  for ( int z = 0 ; z < cycles ; z++ )
  {
    digitalWrite(LED, HIGH);
    delay(del);
    digitalWrite(LED, LOW);
    delay(del);
  }
}

❶ void loop()
{
  blinks = checkTemp();
  blinkLED(blinks, 500);
  delay(2000);
}
```

Основные задачи решаются нашими собственными функциями, поэтому нам остается только вызвать их внутри `void loop()` ❶ и выполнить паузу. Функция `checkTemp()` возвращает целое число, которое сохраняется в переменной `blinks`,

затем `blinkLED()` мигает светодиодом `blinks` раз с задержкой 500 миллисекунд. Перед повторением скетч приостанавливается на 2 секунды.

Загрузите скетч в плату и понаблюдайте за поведением светодиода. Как уже советовалось раньше, попробуйте изменить температуру датчика, обдувая его или зажав между пальцами. Не разбирайте пока схему, она пригодится нам в следующих примерах.

Отображение данных из Arduino на мониторе последовательного порта

До сих пор мы загружали скетчи в Arduino и для вывода информации (например, температуры или направления движения через мост) использовали светодиоды. Мигание светодиодами упрощает организацию обратной связи с Arduino, но вспышки света не слишком информативны. В этом разделе вы узнаете, как использовать проводное соединение Arduino с компьютером и окно монитора последовательного порта в IDE для вывода данных с платы и их отправки на плату с клавиатуры.

Монитор последовательного порта

Чтобы открыть монитор порта, запустите IDE и нажмите Serial Monitor (Монитор порта) на панели инструментов с рис. 5.2. Откроется окно монитора порта (рис. 5.3).

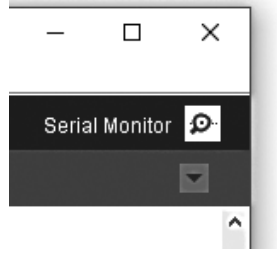


Рис. 5.2. Пиктограмма Serial Monitor (Монитор порта) на панели инструментов

Как показано на рис. 5.3, вверху окна монитора порта есть однострочное поле ввода с кнопкой Send (Отправить) и область вывода под ним, где отображаются данные, полученные с платы Arduino. Если поставить флажок Autoscroll (Автопрокрутка), в области вывода будут отображаться самые свежие данные, а при ее переполнении старые будут «уходить» за границы области. Сняв флажок Autoscroll (Автопрокрутка), вы сможете вручную прокручивать область вывода, чтобы посмотреть все данные.

Рис. 5.3. Окно монитора последовательного порта

Инициализация обмена с монитором порта

До начала использования монитора порта его нужно активировать, добавив следующий вызов в функцию `void setup()`:

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

Число `9600` — это скорость передачи данных между компьютером и платой Arduino, измеряемая в *бодах*. Это число должно соответствовать значению, выбранному в раскрывающемся списке справа внизу в окне монитора порта (см. рис. 5.3).

Отправка текста в монитор порта

Послать текст для отображения в области вывода в мониторе порта можно вызовом функции `Serial.print`:

```
Serial.print("Arduino for Everyone!");
```

Она отправит туда текст в кавычках.

Для вывода текста и принудительного перехода на новую строку воспользуйтесь функцией `Serial.println`:

```
Serial.println("Arduino for Everyone!");
```

Вывод содержимого переменных

Точно так же можно совершить и вывод в монитор порта содержимого переменных. Дальше показано, как вывести содержимое переменной `results`:

```
Serial.println(results);
```

Значения переменных типа `float` по умолчанию выводятся с двумя знаками после десятичной точки. Их количество можно изменить, передав число от 0 до 6 во втором параметре после имени переменной. Чтобы вывести значение вещественной переменной `results` с четырьмя десятичными знаками, выполните следующий вызов:

```
Serial.print(results,4);
```

Проект 12: отображение температуры на мониторе порта

Используя оборудование для проекта 8, выведем температуру в градусах Цельсия и Фаренгейта в окно монитора порта. Для этого мы создадим одну функцию, определяющую температуру, и еще одну — отображающую ее на мониторе порта.

Введите следующий код в IDE:

```
// Проект 12 – отображение температуры на мониторе порта

float celsius = 0;
float fahrenheit = 0;

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

❶ void findTemps()
{
  float voltage = 0;
  float sensor = 0;

  // Прочитать значение с датчика и преобразовать
  // его в градусы Цельсия и Фаренгейта
  sensor = analogRead(0);
  voltage = (sensor*5000)/1024; // Преобразовать в милливольты
  voltage = voltage - 500;      // Учесть смещение
  celsius = voltage / 10;       // Преобразовать милливольты
                                // в градусы Цельсия
  fahrenheit = (1.8 * celsius) + 32; // Преобразовать градусы Цельсия
                                     // в градусы Фаренгейта
}

❷ void displayTemps()
{
  Serial.print("Temperature is ");
  Serial.print(celsius, 2);
  Serial.print(" deg. C / ");
  Serial.print(fahrenheit, 2);
  Serial.println(" deg. F");
  // Здесь используется .println, чтобы вывод следующего
  // замера начинался с новой строки
}

void loop()
{
  findTemps();
  displayTemps();
  delay(1000);
}
```

В этом скетче много разных действий, но мы написали две функции, `findTemps()` ❶ и `displayTemps()` ❷, чтобы упростить его. Они вызываются внутри `void loop()`, которая получилась совсем несложной. На этом примере видно одну из причин создания собственных функций: чтобы сделать скетчи проще и понятнее, а код более модульным и пригодным для многократного использования.

Загрузив скетч, подождите несколько секунд и откройте окно монитора последовательного порта. В области вывода появится список результатов измерения температуры (рис. 5.4).

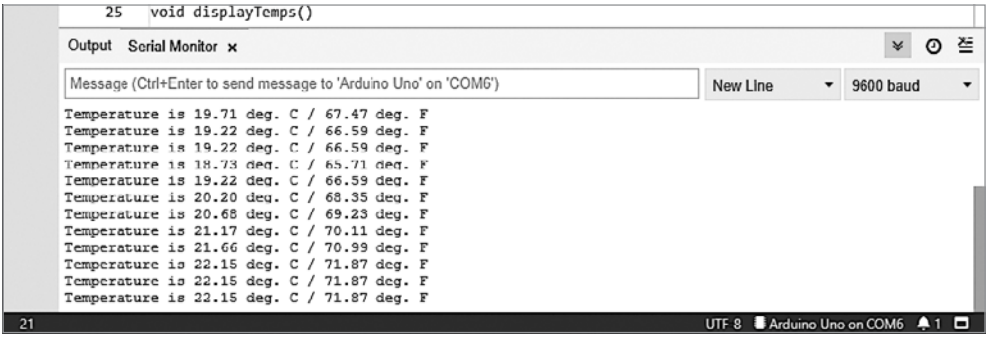


Рис. 5.4. Результаты работы проекта 12

Отладка при помощи монитора порта

Монитор порта можно успешно использовать для *отладки* (поиска и исправления ошибок) скетчей. Если добавить в скетч несколько инструкций `Serial.println();` для вывода кратких примечаний об их местоположении в скетче, можно будет увидеть, когда Arduino выполняет каждую из добавленных инструкций. Например, следующую строку:

```
Serial.println("now in findTemps()");
```

можно вставить в функцию `findTemps()`. Так мы увидим, когда Arduino выполняет эту конкретную функцию.

Принятие решений с помощью инструкций while

Инструкции `while()` используются в скетчах для многократного выполнения фрагментов кода, пока (while) заданное условие остается истинным.

while

Условие в инструкции `while` всегда проверяется *перед* выполнением фрагмента кода. Например, `while ( temperature > 30 )` сначала убедится, что значение `temperature` больше `30`. В условных выражениях внутри круглых скобок можно использовать любые операторы сравнения.

В следующем листинге Arduino сначала отсчитает 10 секунд, а потом продолжит выполнять остальную часть программы:

```
int a = 0; // целое число
while ( a < 10 )
{
  a = a + 1;
  delay(1000);
}
```

Этот скетч сначала присваивает переменной `a` значение `0`. Затем сравнивает `a` с числом `10` (`while ( a < 10 )`), и если ее значение меньше, то прибавляет к ней `1`, ждет секунду (`delay(1000)`) и снова переходит к проверке. Этот процесс повторяется, пока значение `a` не станет больше или равно 10. Как только переменная `a` получит значение `10`, операция сравнения в инструкции `while` вернет `false` и Arduino продолжит выполнение скетча, начиная с инструкции, следующей за фигурной скобкой после цикла `while`.

do-while

В отличие от `while` конструкция `do-while` выполняет проверку *после* выполнения фрагмента кода внутри нее:

```
int a = 0; // целое число
do
{
  delay(1000);
  a = a + 1;
} while ( a < 100 );
```

Здесь код в фигурных скобках будет выполняться *перед* проверкой условия (`while ( a < 100 )`). То есть, даже если условие не выполняется, цикл будет выполнен не менее одного раза. Вам самим придется решить, какую инструкцию выбрать — `while` или `do-while` — в зависимости от требований конкретного проекта.

Передача данных из монитора порта в Arduino

Чтобы послать данные из монитора порта в Arduino, нужно, чтобы она извлекала полученные данные из *буфера последовательного порта*. Это составляющая Arduino, которая принимает данные извне, полученные через выводы последовательного порта (цифровые входы/выходы 0 и 1), связанные с компьютером через интерфейс USB. В последовательном буфере хранятся входящие данные, введенные в окне монитора порта и отправленные пользователем.

Проект 13: умножение числа на два

Чтобы увидеть процесс передачи и приема данных через монитор порта, рассмотрим следующий скетч. Он принимает одну цифру от пользователя, умножает ее на 2 и отправляет результат на монитор порта. Загрузите скетч в плату, откройте окно Serial Monitor (Монитор порта), выберите No Line Ending (Нет конца строки) в раскрывающемся списке внизу. Чтобы отправить введенные данные в Arduino, нужно нажать комбинацию Ctrl+Enter (не просто Enter).

```
// Проект 13 — умножение числа на два

int number;

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

void loop()
{
  number = 0;     // Обнулить переменную, подготовив
                  // ее к приему нового числа
  Serial.flush(); // Очистить буфер порта от мусора перед ожиданием
❶ while (Serial.available() == 0)
  {
    // Ничего не делать, пока что-то не появится в буфере порта
  }
❷ while (Serial.available() > 0)
  {
    number = Serial.read() - '0';
    // Прочитать цифру из буфера порта,
    // вычесть из ASCII-кода цифры
    // код символа '0', чтобы получить число
  }

  // Вывести число!
  Serial.print("You entered: ");
  Serial.println(number);
  Serial.print(number);
  Serial.print(" multiplied by two is ");
  number = number * 2;
  Serial.println(number);
}
```

Функция `Serial.available()` в первой инструкции `while` ❶ возвращает `0`, если пользователь ничего не ввел в монитор порта. То есть она сообщает Arduino: «Ничего не делай, пока пользователь не введет что-нибудь». Следующая инструкция `while` ❷

извлекает цифру из буфера порта и преобразует код символа, обозначающего цифру, в фактическое целое число. Дальше Arduino выводит число, полученное из буфера порта, и результат его умножения на 2.

Функция `Serial.flush()` в начале скетча очищает буфер порта от любых данных, которые могли там находиться, готовя скетч к приему следующих данных. На рис. 5.5 изображено окно монитора порта после запуска скетча.

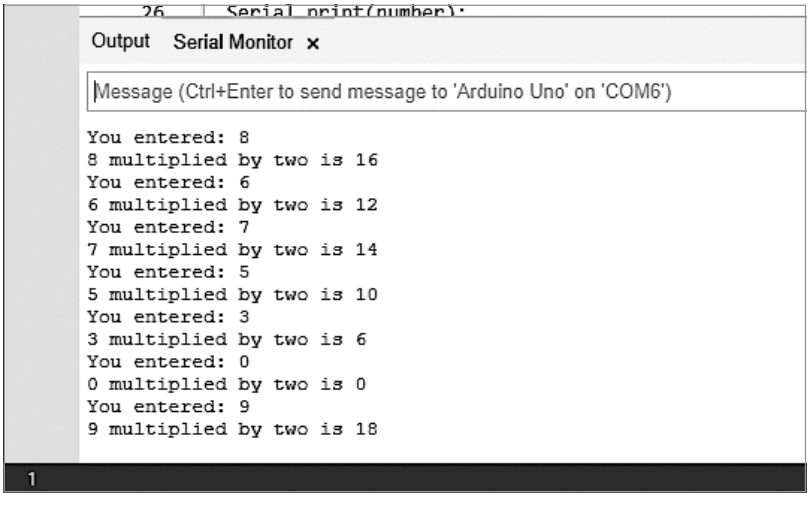


Рис. 5.5. Пример ввода и вывода в проекте 13

Несмотря на то что теперь вы можете вводить данные в мониторе порта для обработки платой Arduino, использование переменных типа `int` сильно ограничивает диапазон допустимых чисел. Чтобы его расширить, используйте переменные типа `long`, о которых рассказывается далее.

Переменные типа long

Чтобы использовать монитор порта для ввода чисел, состоящих более чем из одной цифры, нужно немного расширить предыдущий скетч, как будет показано ниже. Но, когда приходится работать с большими числами, диапазон значений, представляемый типом `int`, может оказаться слишком узким — переменные этого типа не способны хранить значения больше 32 767. К счастью, это ограничение легко преодолеть с помощью переменных типа `long`. Они могут хранить целые числа от –2 147 483 648 до 2 147 483 647. Этот диапазон намного шире, чем в `int` (от –32 768 до 32 767).

Проект 14: использование переменных типа long

В этом проекте мы используем монитор порта для передачи значений типа `long` и чисел, состоящих из нескольких цифр. Этот скетч принимает наше число, умножает его на 2 и выводит результат в монитор порта.

```
// Проект 14 – использование переменных типа long

long number = 0;
long a = 0;

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

void loop()
{
  number = 0;     // Обнулить переменную, подготовив
                  // ее к приему нового числа
  Serial.flush(); // Очистить буфер порта от мусора перед ожиданием
  while (Serial.available() == 0)
  {
    // Ничего не делать, пока что-то не появится в буфере порта,
    // когда что-то появится в буфере, Serial.available вернет
    // количество символов в буфере, ожидающих обработки
  }

  // Как минимум один символ имеется в буфере,
  // начать вычисления
  while (Serial.available() > 0)
  {
    // Сдвинуть предыдущие цифры на разряд влево;
    // иными словами, 1 превратится в 10, если в буфере имеются данные
    number = number * 10;

    // Прочитать следующую цифру из буфера и вычесть из нее
    // код символа '0', чтобы превратить в фактическое целое число
    a = Serial.read() - '0';

    // Прибавить это значение к накапливаемому значению
    number = number + a;

    // Выполнить короткую задержку, чтобы дать возможность
    // следующим цифрам достичь буфера
    delay(5);
  }
  Serial.print("You entered: ");
  Serial.println(number);
  Serial.print(number);
```

```
  Serial.print(" multiplied by two is ");
  number = number * 2;
  Serial.println(number);
}
```

В этом примере два цикла `while` позволяют Arduino принять несколько цифр из монитора порта. Когда в буфер поступает первая цифра (самый левый разряд во введенном числе), она преобразуется в число и прибавляется к переменной `number`. Если пользователь ввел всего одну цифру, скетч завершит цикл и продолжит выполнение. Если была введена еще одна цифра (например, `2` в числе `42`), значение `number` будет умножено на 10, чтобы сдвинуть первую на один разряд влево, и к значению `number` добавится новая цифра. Этот цикл продолжается, пока в `number` не будет добавлена самая правая цифра во введенном числе. Не забудьте выбрать пункт No line ending (Нет конца строки) в раскрывающемся списке (рис. 5.6) в нижней части окна монитора порта.

На рис. 5.6 изображено окно монитора порта после запуска скетча.

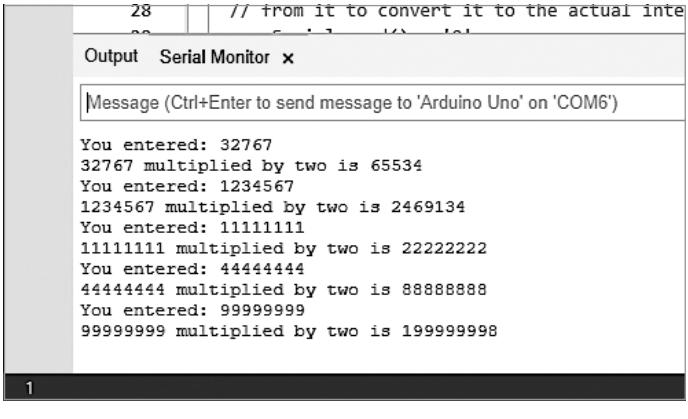


Рис. 5.6. Пример ввода и вывода в проекте 14

Что дальше?

Умение писать свои функции — очень важный навык, который помогает упростить скетчи и сэкономить время и силы. Достойное применение полученным здесь знаниям мы найдем в следующей главе. Там вы познакомитесь с другими математическими возможностями Arduino и перейдете к программированию игр.

6

Числа, переменные и арифметика

В этой главе вы:

- научитесь генерировать случайные числа;
- создадите электронный игровой кубик;
- познакомитесь с двоичной системой счисления;
- освоите работу с микросхемой сдвигового регистра, позволяющей увеличить количество цифровых входов;
- проверите свои знания двоичной арифметики в небольшой викторине;
- узнаете о массивах переменных;
- научитесь выводить цифры на семисегментный индикатор;
- познакомитесь с математической функцией деления по модулю;
- создадите цифровой термометр.

В этой главе вас ждет знакомство с новыми функциями, которые помогут создавать новые виды проектов — генерация случайных чисел, новые математические функции и хранилища переменных в виде упорядоченных списков *(массивов)*. Вы узнаете, как использовать цифровые светодиодные модули для отображения данных и простых изображений. Полученные умения мы используем для создания игры, цифрового термометра и других устройств.

Случайные числа

Возможность генерировать случайные числа с помощью программы очень пригодится в проектах игр и для создания разных эффектов. К примеру, в реализации игрового кубика или лотереи на основе Arduino, для создания световых эффектов с помощью светодиодов или звуковых и световых эффектов в игре-викторине.

К сожалению, сама плата Arduino не может воспроизводить по-настоящему случайные числа. Но можно помочь ей, выбирая произвольное начальное число, которое потом будет использоваться в вычислениях других случайных чисел.

Использование электрического поля для генерации случайных чисел

Самый простой способ сгенерировать случайное число в Arduino — написать программу, которая будет читать напряжение со свободного (ни к чему не подключенного) аналогового входа (например, с нулевого) в функции `void setup()`:

```
randomSeed(analogRead(0));
```

Даже если к этому входу ничего не подключено, на нем все равно есть некоторое измеримое статическое напряжение, создаваемое извне. Его величина изменяется хаотически. Мы можем сохранить результат измерения этого наведенного электрического потенциала в целочисленной переменной и сделать его начальным значением для вычисления последовательности псевдослучайных чисел функцией `random(lower, upper)`. Параметры *`lower`* и *`upper`* определяют нижнюю и верхнюю границы диапазона случайных чисел. Вот как можно сгенерировать случайное число между 100 и 1000:

```
int a = 0;
a = random(100, 1001);
```

Здесь число 1001 (а не 1000) — верхняя граница, потому что значение этой границы *не входит* в диапазон.

С другой стороны, для генерации случайного числа от 0 до некоторого значения достаточно передать только верхнюю границу. Следующая инструкция сгенерирует случайное число в диапазоне от 0 до 6 (включительно):

```
a = random(7);
```

Пример скетча в листинге 6.1 генерирует случайные числа между 0 и 1000, а также между 10 и 50.

Листинг 6.1. Генератор случайных чисел

```
int r = 0;
void setup()
{
  randomSeed(analogRead(0));
  Serial.begin(9600);
}

void loop()
```

```
{
  Serial.print("Random number between zero and 1000 is: ");
  r = random(0, 1001);
  Serial.println(r);
  Serial.print("Random number between ten and fifty is: ");
  r = random(10, 51);
  Serial.println(r);
  delay(1000);
}
```

На рис. 6.1 показаны результаты, отображаемые в окне монитора последовательного порта.

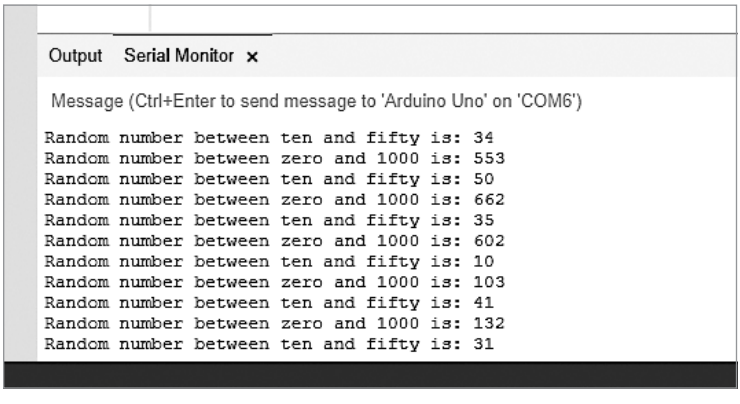


Рис. 6.1. Вывод скетча из листинга 6.1

Теперь применим знания о генерации случайных чисел для создания электронного кубика.

Проект 15: электронный кубик

Наша цель: случайно выбрать и зажечь один из шести светодиодов для имитации броска игрового кубика. Мы будем генерировать случайное число от 1 до 6 и включать соответствующий светодиод, чтобы показать результат. Для этого создадим функцию, случайно выбирающую один из шести светодиодов и оставляющую его включенным на некоторое время. После включения или сброса платы Arduino со скетчем она должна начать быстро зажигать случайно выбранный светодиод, создавая эффект паузы-размышления, оставлять его включенным на некоторое время, потом постепенно уменьшить скорость выбора и, наконец, остановиться на одном из светодиодов. Он и останется гореть, пока плата не будет сброшена или выключена.

Оборудование

Чтобы сделать кубик, нам нужно следующее:

- шесть светодиодов любого цвета (LED1–LED6);
- один резистор номиналом 560 Ом (R1);
- несколько отрезков провода разной длины;
- одна макетная плата среднего размера;
- плата Arduino и кабель USB.

Схема

Поскольку в каждый момент времени включаться будет только один светодиод, мы соединим катоды всех с выводом GND через единственный ограничивающий резистор. Принципиальная схема кубика показана на рис. 6.2.

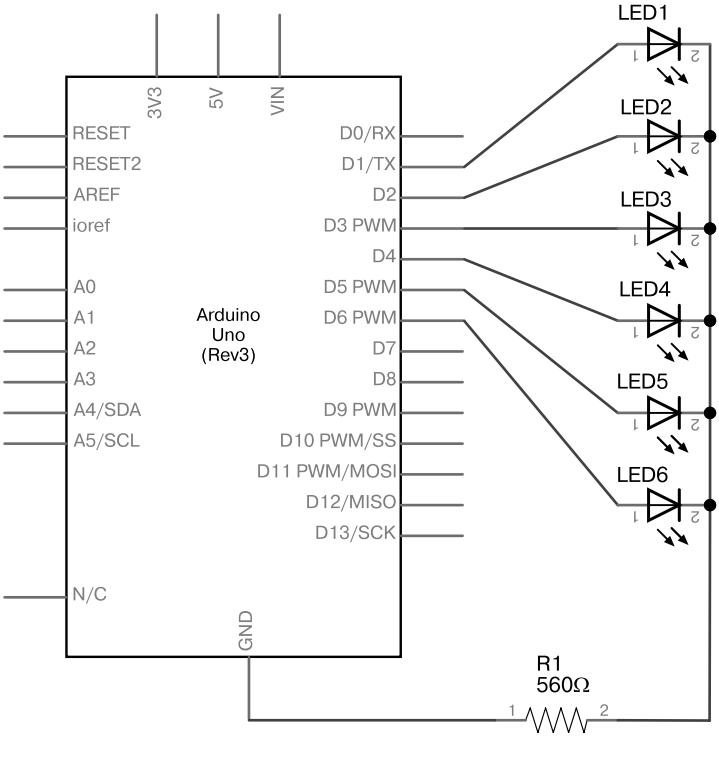


Рис. 6.2. Принципиальная схема проекта 15

Скетч

Далее приводится исходный код скетча, реализующего работу кубика:

```
// Проект 15 — электронный кубик

void setup()
{
  randomSeed(analogRead(0));      // Начальное число
                                  // для генератора случайных чисел
  for ( int z = 1 ; z < 7 ; z++ ) // Настроить контакты 1–6
  {                               // на работу в режиме выходов
    pinMode(z, OUTPUT);
  }
}

void randomLED(int del)
{
  int r;
  r = random(1, 7);      // Получить случайное число от 1 до 6
  digitalWrite(r, HIGH); // Включить соответствующий светодиод
  if (del > 0)
  {
❶     delay(del);        // Оставить включенным в течение
  }                      // указанного времени
❷ else if (del == 0)
  {
    do                   // Если указана нулевая задержка, оставить
    {}                   // светодиод включенным "навсегда"
❸   while (1);
  }
  digitalWrite(r, LOW);  // Выключить светодиод
}

void loop()
{
  int a;
  // Создать эффект выбора светодиода
  for ( a = 0 ; a < 100 ; a++ )
  {
    randomLED(50);
  }
  // Постепенно замедлить эффект
❹ for ( a = 1 ; a <= 10 ; a++ )
  {
    randomLED(a * 100);
  }
  // И остановиться на случайно выбранном светодиоде
  randomLED(0);
}
```

Здесь в функции `void setup()` для настройки выводов на работу в режиме цифровых выходов использован цикл. `randomLED()` принимает целое число из вызова функции `delay()` ❶, чтобы оставить светодиод включенным на некоторое время. Если функция получит значение задержки `0` ❷, она оставит светодиод включенным «навсегда», войдя в бесконечный цикл ❸

```
do {} while (1);
```

потому что 1 — всегда 1.

Чтобы «бросить кубик», нужно выполнить сброс платы Arduino. Теперь нужно создать эффект размышлений с постепенным замедлением перед остановкой на окончательно выбранном светодиоде. Для этого сначала 100 раз включается случайно выбранный светодиод с задержкой на 50 миллисекунд ❹. Затем скорость перебора уменьшается за счет увеличения периода, когда светодиод остается включенным, со 100 до 1000 миллисекунд и с шагом 100 миллисекунд. Цель этого решения — сымитировать замедление вращения кубика перед полной остановкой, после чего Arduino отображает результат выпадения кубика, включая один из светодиодов в последней строке:

```
randomLED(0);
```

Доработка скетча

Этот проект можно изменять. Например, можно добавить еще шесть светодиодов для имитации броска сразу двух кубиков или для отображения результата единственным встроенным светодиодом, мигая им выпавшее число раз. Можно добавить кнопку, нажатием которой выполняется бросок. Поэкспериментируйте, используя свое воображение и полученные знания!

Краткое введение в двоичную систему счисления

Многие дети учатся счету на базе десятичной системы счисления, но компьютеры (и плата Arduino) производят вычисления с применением двоичной.

Двоичные числа

Двоичные числа состоят только из нулей и единиц — например, 10101010. Каждая цифра в двоичном числе справа налево представляет число 2 в степени, соответствующей порядковому (начиная с нуля) номеру разряда (счет разрядов ведется справа налево). Сумма произведений в каждом разряде определяет величину значения числа.

Рассмотрим двоичное число 10101010 (табл. 6.1). Чтобы преобразовать его в десятичную систему счисления, сложим степени двойки во всех разрядах, как показано в последней строке табл. 6.1:

$$128 + 0 + 32 + 0 + 8 + 0 + 2 + 0.$$

Получится десятичное число 170. То есть двоичное число равно десятичному 170. Двоичное число с восемью разрядами (или *битами*) представляет 1 *байт* данных. Байт может иметь числовое значение от 0 до 255. Самый левый бит называют *старшим* (Most Significant Bit, MSB), а самый правый — *младшим* (Least Significant Bit, LSB).

Двоичные числа отлично подходят для хранения некоторых типов данных (признаки «включено/выключено» для светодиодов, «да/нет» для параметров настройки и состояния цифровых выходов). Двоичные числа — это строительные блоки всех типов компьютерных данных.

Таблица 6.1. Пример преобразования двоичного числа в десятичное

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	1	0	1	0	1	0	Двоичное
128	64	32	16	8	4	2	1	Десятичное

Переменные типа byte

Переменные типа `byte` — один из способов хранения двоичных чисел. Создать переменную этого типа можно так:

```
byte outputs = B11111111;
```

Буква `B` в начале числа сообщает, что следующее за ней число должно интерпретироваться как двоичное (в данном случае 11111111), а не десятичное (листинг 6.2).

Листинг 6.2. Демонстрация двоичного числа

```
byte a;

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

void loop()
{
  for ( int count = 0 ; count < 256 ; count++ )
  {
```

```
      a = count;
      Serial.print("Base-10 = ");
❶     Serial.print(a, DEC);
      Serial.print(" Binary = ");
❷     Serial.println(a, BIN);
      delay(1000);
    }
  }
```

Скетч выводит значение переменной типа `byte` вызовом функции `Serial.print()` в виде десятичного числа при использовании параметра `DEC` ❶ и в виде двоичного при использовании `BIN` ❷. После загрузки скетча в плату в окне монитора порта должен появиться вывод (рис. 6.3).

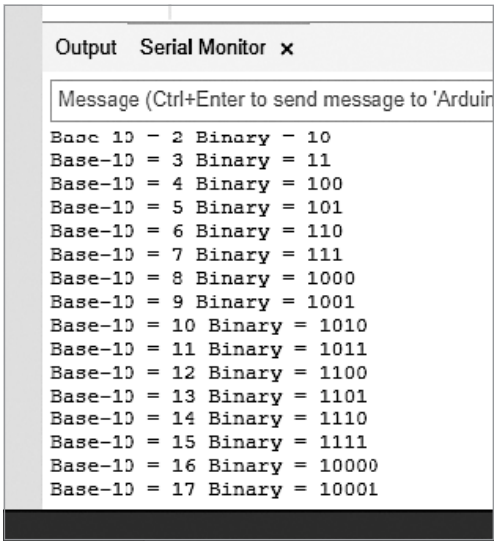


Рис. 6.3. Вывод скетча из листинга 6.2

Увеличение числа цифровых выходов с применением сдвигового регистра

В Arduino Uno 13 выводов, которые можно использовать как выходы. Но иногда их может быть недостаточно. С помощью *сдвигового регистра* можно добавить дополнительные цифровые выходы для проекта. Сдвиговый регистр — это микросхема с восемью цифровыми выходами, которыми можно управлять, посылая в микросхему байты данных. В наших проектах мы используем сдвиговый регистр 74HC595, изображенный на рис. 6.4.

Рис. 6.4. Сдвиговый регистр 74HC595

У сдвигового регистра 74HC595 восемь цифровых выходов, действующих подобно тем, что есть на плате Arduino. Для управления сдвиговым регистром нужно задействовать три цифровых выхода платы, в итоге мы получим пять дополнительных цифровых выходов.

Принцип действия этого регистра прост: мы посылаем в него 1 байт данных (8 бит), а он включает или выключает восемь своих выходов согласно состояниям разрядов в этом байте. Биты в байте данных соответствуют цифровым выходам регистра в порядке от старшего к младшему. Крайнему левому биту соответствует выход 7 сдвигового регистра, а самому правому — выход 0. Если послать в сдвиговый регистр байт `B10000110`, он включит напряжение на выходах 7, 2 и 1 и выключит на выходах 0 и 3–6. Эти уровни напряжения на выходах останутся, пока регистр не получит другой байт данных или не будет отключено напряжение питания.

Одновременно к одним и тем же трем цифровым выходам на плате можно подключить несколько сдвиговых регистров и с каждым из них получить восемь дополнительных цифровых выходов. Это очень удобно, когда нужно управлять большим количеством светодиодов. Давайте теперь попрактикуем такое управление, создав «дисплей» для вывода двоичных чисел.

Проект 16: светодиодный индикатор для двоичных чисел

В этом проекте мы реализуем отображение двоичных чисел от 0 до 255 с помощью восьми светодиодов. Скетч будет перебирать в цикле числа от 0 до 255 и посылать каждое из них в сдвиговый регистр, управляющий светодиодами для отображения двоичных эквивалентов.

Оборудование

Ниже перечислено необходимое оборудование:

- один сдвиговый регистр 74HC595;
- восемь светодиодов (LED1–LED8);
- восемь резисторов номиналом 560 Ом (R1–R8);
- одна макетная плата;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Подключение микросхемы 74HC595

На рис. 6.5 показано отображение регистра 74HC595 на принципиальных схемах.

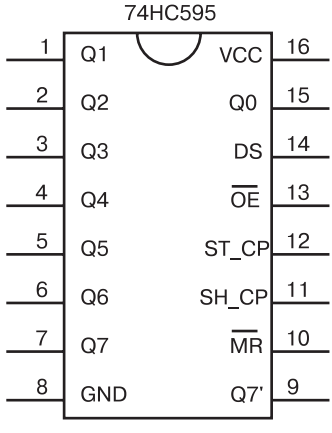


Рис. 6.5. Изображение сдвигового регистра 74HC595 на принципиальных схемах

У микросхемы сдвигового регистра всего 16 выводов:

- выводы 15 и с 1 по 7 — это восемь цифровых выходов, которыми мы будем управлять (отмечены как Q0–Q7);
- вывод Q7 соответствует первому биту, посылаемому в сдвиговый регистр, а вывод Q0 — последнему;
- вывод 8 подключается к «земле» (GND);
- вывод 9 — «выходные данные», используется для передачи данных другому сдвиговому регистру при его наличии;
- вывод 10 всегда должен быть подключен к шине питания 5 В (например, к контакту 5 V на плате Arduino);
- выводы 11 и 12 называют *входом тактовых импульсов* и *защелкой*;
- вывод 13 используется для разрешения выхода (переключения выходов между высокоимпедансным и рабочим состоянием). Обычно подключается к «земле» (GND);
- вывод 14 — вход для битов данных, последовательно посылаемых платой Arduino;
- вывод 16 — питание 5 В (подключается к контакту 5 V на плате Arduino).

Ориентация контактов определяется по полукруглой метке на корпусе микросхемы сдвигового регистра (рис. 6.4). Она находится между выводами 1 и 16.

Выводы нумеруются в направлении против часовой стрелки (рис. 6.6).

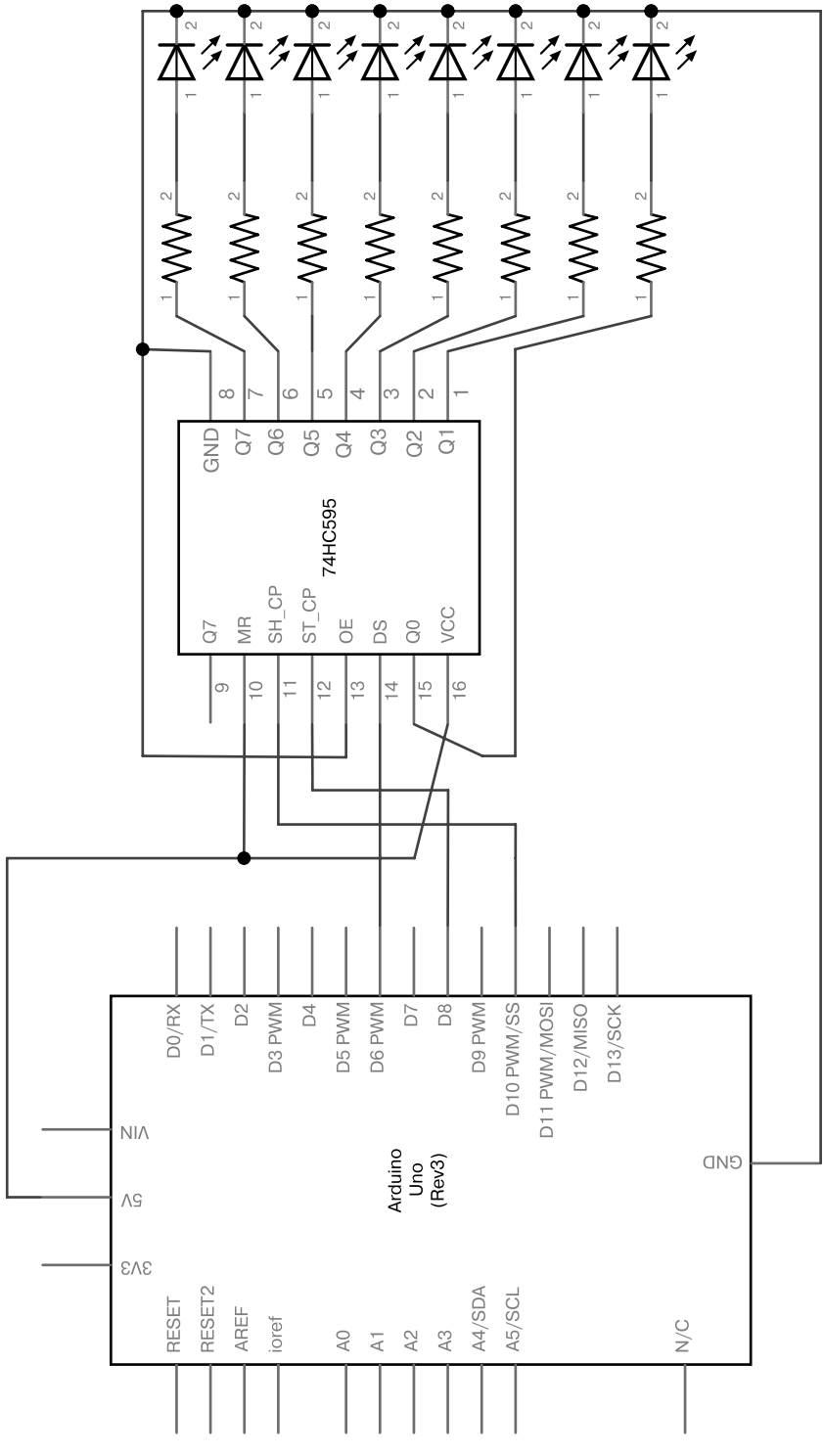


Рис. 6.6. Принципиальная схема для проекта 16

ПРИМЕЧАНИЕ

Не разбирайте собранную схему после окончания проекта. Она еще пригодится дальше.

Скетч

А теперь скетч:

```
// Проект 16 — светодиодный индикатор для двоичных чисел

#define DATA 6   // Цифровой выход 6 — к выводу 14 микросхемы 74HC595
#define LATCH 8  // Цифровой выход 8 — к выводу 12 микросхемы 74HC595
#define CLOCK 10 // Цифровой выход 10 — к выводу 11 микросхемы 74HC595

void setup()
{
  pinMode(LATCH, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(DATA, OUTPUT);
}

void loop()
{
  int i;
  for ( i = 0; i < 256; i++ )
  {
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, MSBFIRST, i);
    digitalWrite(LATCH, HIGH);
    delay(200);
  }
}
```

Три контакта на плате, к которым подключается сдвиговый регистр, мы настроили в функции `void setup()` на работу в режиме цифровых выходов и добавили в `void loop()` цикл, который последовательно перебирает числа от 0 до 255. Все волшебство здесь в цикле. Когда цикл `for` выводит байт данных (например, 240 или B11110000) в сдвиговый регистр, происходят три события:

- на вывод 12 (защелка) микросхемы подается уровень `LOW` (сигнал низкого уровня выводится на выход 8 платы Arduino). Это подготавливает сдвиговый регистр к подаче на вывод 12 микросхемы уровня `HIGH`, который «защелкнет» данные на ее выходах, как только `shiftOut()` завершит свою работу;
- байт данных (например, B11110000) посылается с цифрового выхода 6 платы Arduino в сдвиговый регистр. При этом функции `shiftOut()` сообщается на-

правление интерпретации байта данных. Например, если передать функции параметр `LSBFIRST`, светодиоды 1–4 включатся, а остальные — выключатся. Если передать параметр `MSBFIRST`, включатся светодиоды 5–8, а остальные — выключатся;

- на вывод 12 микросхемы подается уровень `HIGH` (5 В). Так сдвиговому регистру сообщается об окончании передачи битов. В этот момент сдвиговый регистр изменит состояния своих выходов в соответствии с полученными данными.

Проект 17: игра «Двоичная викторина»

В этом проекте мы создадим игру «Двоичная викторина» при помощи случайных чисел, монитора последовательного порта и схемы из проекта 16. Плата Arduino будет отображать светодиодами случайное двоичное число, а вы должны ввести его в десятичном виде в окне монитора порта. Затем монитор сообщит, верно ли введенное вами число, и игра отобразит на светодиодах новое число.

Алгоритм

Алгоритм можно поделить на три функции. `displayNumber()` отобразит двоичное число с помощью светодиодов. `getAnswer()` примет число из монитора порта и выведет его обратно. `checkAnswer()` сравнит введенное число с отображаемым на светодиодах, сообщит, правильно ли пользователь решил задачу, и отобразит правильный ответ в случае ошибки.

Скетч

Скетч генерирует случайные числа от 0 до 255, выводит их в двоичном виде с использованием светодиодов, предлагает участнику ввести это число в десятичном виде и отображает результаты в окне монитора порта. Вы уже знакомы с функциями из скетча, поэтому, несмотря на большой объем, прочесть код должно быть несложно. Исследовать его помогут комментарии в скетче и дальнейшее описание.

```
// Проект 17 – игра "Двоичная викторина"

#define DATA   6 // К выводу 14 микросхемы 74HC595
#define LATCH  8 // К выводу 12 микросхемы 74HC595
#define CLOCK 10 // К выводу 11 микросхемы 74HC595

int number = 0;
int answer = 0;
```

```
❶ void setup()
  {
    pinMode(LATCH, OUTPUT);    // Подготовить 74HC595
    pinMode(CLOCK, OUTPUT);
    pinMode(DATA, OUTPUT);
    Serial.begin(9600);
    randomSeed(analogRead(0)); // Инициализировать генератор
                               // случайных чисел
    displayNumber(0);          // Выключить все светодиоды
  }
❷ void displayNumber(byte a)
  {
    // Посылает байт для отображения на светодиодах
    digitalWrite(LATCH, LOW);

    shiftOut(DATA, CLOCK, MSBFIRST, a);
    digitalWrite(LATCH, HIGH);
  }
❸ void getAnswer()
  {
    // Принимает ответ игрока
    int z = 0;
    Serial.flush();
    while (Serial.available() == 0)
    {
      // Ничего не делать, пока что-то не появится в буфере порта
    }
    // Как минимум один символ имеется в буфере,
    // начать вычисления
    while (Serial.available() > 0)
    {
      // Сдвинуть предыдущие цифры на разряд влево;
      // иными словами, 1 превратится в 10, если в буфере есть данные
      answer = answer * 10;

      // Прочитать следующую цифру из буфера и вычесть из нее
      // код символа '0', чтобы превратить в фактическое целое число
      z = Serial.read() - '0';

      // Прибавить это значение к накапливаемому значению
      answer = answer + z;

      // Выполнить короткую задержку, чтобы дать возможность
      // следующим цифрам достичь буфера
      delay(5);
    }
    Serial.print("You entered: ");
    Serial.println(answer);
  }
```

```
❹ void checkAnswer()
  {
    // Проверяет ответ игрока и выводит результаты
    if (answer == number) // Верно!
    {
      Serial.print("Correct! ");
      Serial.print(answer, BIN);
      Serial.print(" equals ");
      Serial.println(number);
      Serial.println();
    }
    else // Неверно
    {
      Serial.print("Incorrect, ");
      Serial.print(number, BIN);
      Serial.print(" equals ");
      Serial.println(number);
      Serial.println();
    }
    answer = 0;
    delay(10000); // Дать игроку время прочитать результаты
  }
❺ void loop()
  {
    number = random(256);
    displayNumber(number);
    Serial.println("What is the binary number in base-10? ");
    getAnswer();
    checkAnswer();
  }
```

Разберемся в работе этого скетча. Функция `void setup()` ❶ настраивает цифровые выходы, к которым подключен сдвиговый регистр, запускает обмен с монитором порта и устанавливает начальное значение для генератора случайных чисел. Функция `displayNumber()` ❷ принимает байт данных и передает его в сдвиговый регистр, к которому подключены светодиоды для отображения байта в двоичной форме (как в проекте 16). Функция `getAnswer()` ❸ извлекает из монитора порта введенное пользователем число (как в проекте 14) и выводит его, как показано на рис. 6.7.

Функция `checkAnswer()` ❹ сравнивает число пользователя, полученное с помощью `getAnswer()`, со случайным, сгенерированным в функции `void loop()`. Дальше игроку сообщается, правильно или неправильно он ответил на вопрос с выводом числа в двоичной и десятичной форме. Наконец, в основной функции `void loop()` ❺ генерируется случайное двоичное число для викторины, затем вызываются функции для его отображения светодиодами, и после этого принимается и проверяется ответ пользователя.

На рис. 6.7 изображено окно монитора порта в ходе игры.

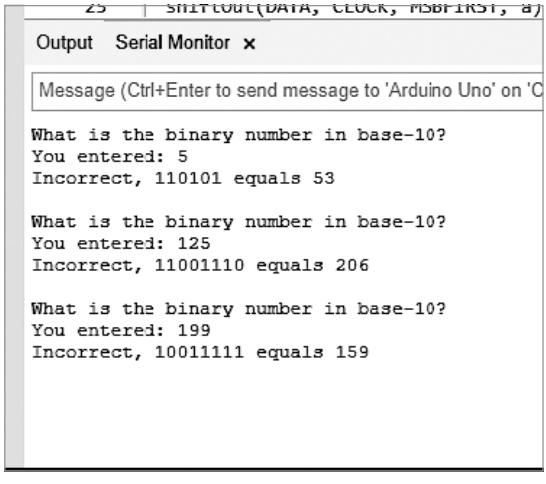


Рис. 6.7. Проект 17 в процессе игры

Массивы

Массив — это множество переменных, или значений, сгруппированных так, что к ним можно обращаться как единому целому. При наличии большого количества однотипных, связанных данных обычно предпочтительнее использовать массивы. Так их можно сохранить организованными.

Определение массива

Составляющие называются *элементами*. Представьте, что есть шесть вещественных переменных, хранящих значения температуры за последние шесть часов. Вместо того чтобы всем им дать уникальные имена, можно определить массив с именем `temperatures`:

```
float temperatures[6];
```

Можно вставить начальные значения элементов массива при его определении. В этом случае размер массива можно не указывать:

```
float temperatures[]={11.1, 12.2, 13.3, 14.4, 15.5, 16.6};
```

Обратите внимание, что в этом случае размер массива в квадратных скобках (`[]`) не указан — он определяется по числу элементов в фигурных скобках (`{}`).

Заметьте также, что, независимо от размеров, массивы могут хранить элементы только одного типа.

Обращение к значениям в массиве

Нумерация элементов в массиве начинается слева и с 0. В массиве `temperatures[]` содержатся элементы с порядковыми номерами от 0 до 5. Обращаться к отдельным элементам массива можно, добавляя порядковый номер элемента в квадратных скобках после имени массива. Например, присвоить первому элементу в массиве `temperatures[]` (имеет текущее значение `11.1`) новое значение `12.34` можно так:

```
temperatures[0] = 12.34;
```

Запись в массивы и чтение из них

В листинге 6.3 показаны приемы записи и чтения значений из массива с пятью элементами. Первый цикл `for` в скетче записывает случайные числа во все элементы массива, а второй `for` извлекает элементы и выводит их на монитор порта.

Листинг 6.3. Демонстрация операций чтения/записи с массивом

```
void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

int array[5]; // Определение массива с пятью элементами
void loop()
{
  int i;
  Serial.println();
  for ( i = 0 ; i < 5 ; i++ ) // Запись в массив
  {
    array[i] = random(10);    // Случайные числа от 0 до 9
  }
  for ( i = 0 ; i < 5 ; i++ ) // Вывести содержимое массива
  {
    Serial.print("array[");
    Serial.print(i);
    Serial.print("] contains ");
    Serial.println(array[i]);
  }
  delay(5000);
}
```

На рис. 6.8 изображено окно монитора порта.

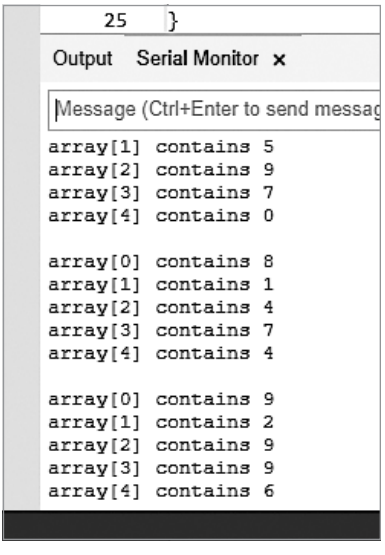


Рис. 6.8. Скетч из листинга 6.3 в действии

Теперь вы знаете, как работать с двоичными числами, сдвиговыми регистрами и массивами. Используем новые знания на практике. В следующем разделе мы подключим к плате несколько цифровых индикаторов.

Семисегментные светодиодные индикаторы

Работать со светодиодами очень интересно, но их можно применять для отображения весьма ограниченного набора данных. В этом разделе мы начнем использовать светодиодные цифровые семисегментные индикаторы (рис. 6.9).

Эти индикаторы отлично подходят для отображения чисел. Именно поэтому они используются в цифровых будильниках, спидометрах и других устройствах для отображения числовой информации. У каждого индикатора есть семисегментный дисплей с восемью светодиодами. Выпускаются индикаторы со светодиодами разного цвета. Для уменьшения числа выводов у индикаторов все аноды или катоды светодиодов соединены с общим выводом, который называют *общим анодом* или *общим катодом* соответственно. В наших проектах будут использоваться индикаторы с общим катодом.

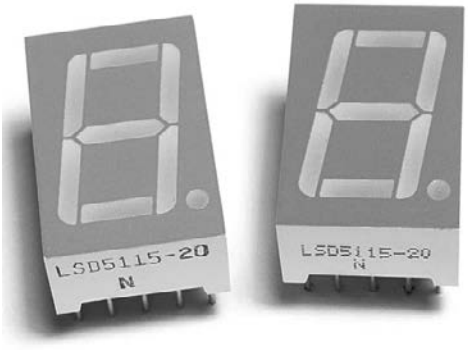


Рис. 6.9. Семисегментные индикаторы

Светодиодные сегменты снабжены метками от *A* до *G*, а также *DP* (Decimal Point — «десятичная точка»). Каждому сегменту соответствует свой вывод, соединенный с анодом, а катоды подключены к общему. Размещение сегментов всегда соответствует изображенному на рис. 6.10, где *A* находится вверху, *B* — справа и т. д. Например, чтобы показать цифру 7, нужно подать напряжение на сегменты *A*, *B* и *C*.

Порядок размещения выводов на корпусе светодиодного индикатора может отличаться у разных производителей, но они всегда соответствуют шаблону с рис. 6.10. Приобретая такие индикаторы, обязательно возьмите схему расположения выводов у продавца, чтобы не терять времени на самостоятельное выяснение.

Для изображения семисегментных индикаторов на принципиальных схемах мы будем использовать значок, показанный на рис. 6.11.

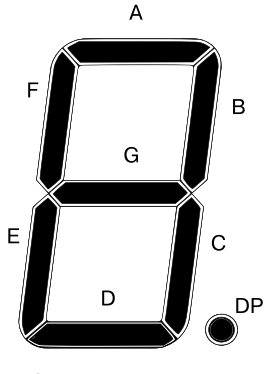


Рис. 6.10. Назначение светодиодов типичного семисегментного индикатора

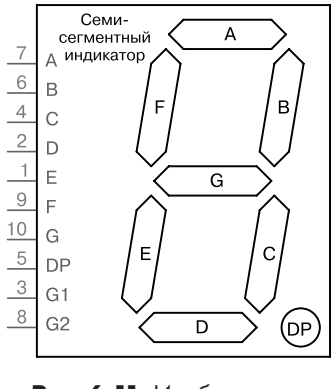


Рис. 6.11. Изображение семисегментного индикатора на принципиальных схемах

Управление сегментами

Для управления сегментами мы воспользуемся методом из проекта 17, подключив выводы с *A* по *DP* к выводам сдвигового регистра *Q0–Q7*. Какие сегменты включать и выключать для отображения цифр или букв, можно определить по матрице (табл. 6.2).

Верхняя строка — это выводы сдвигового регистра, управляющие сегментами. Их названия перечислены во второй строке. Все остальные строки соответствуют отображаемым цифрам и содержат значения в двоичном и десятичном виде, которые нужно послать в сдвиговый регистр.

Таблица 6.2. Матрица отображения цифр на семисегментном индикаторе

Выводы сдвигового регистра	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	
Сегмент	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**DP**	**Десятичное**
0	1	1	1	1	1	1	0	0	252
1	0	1	1	0	0	0	0	0	96
2	1	1	0	1	1	0	1	0	218
3	1	1	1	1	0	0	1	0	242
4	0	1	1	0	0	1	1	0	102
5	1	0	1	1	0	1	1	0	182
6	1	0	1	1	1	1	1	0	190
7	1	1	1	0	0	0	0	0	224
8	1	1	1	1	1	1	1	0	254
9	1	1	1	1	0	1	1	0	246
A	1	1	1	0	1	1	1	0	238
B	0	0	1	1	1	1	1	0	62
C	1	0	0	1	1	1	0	0	156
D	0	1	1	1	1	0	1	0	122
E	1	0	0	1	1	1	1	0	158
F	1	0	0	0	1	1	1	0	142

Например, чтобы отобразить цифру 7, как показано на рис. 6.12, нужно включить сегменты *A*, *B* и *C*, которым соответствуют выводы Q0, Q1 и Q2 сдвигового регистра. То есть, чтобы установить высокий уровень на первых трех выводах, соответствующих перечисленным сегментам, мы должны послать в сдвиговый регистр байт B1110000 (вызовом `shiftOut` с параметром `LSBFIRST`).

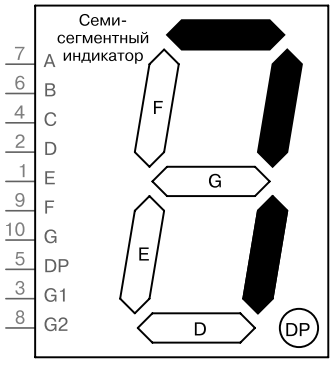


Рис. 6.12. Отображение цифры *7*

В следующем проекте соберем схему, которая последовательно будет отображать цифры от 0 до 9 и буквы от A до F. Потом цикл повторится, но с включенным сегментом десятичной точки.

Проект 18: дисплей с одной цифрой

В этом проекте мы соберем схему, использующую один цифровой индикатор.

Оборудование

Ниже перечислено необходимое оборудование:

- один сдвиговый регистр 74HC595;
- один семисегментный светодиодный индикатор с общим катодом;
- один резистор номиналом 560 Ом (R1);
- одна большая макетная плата;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Схема

Схема для проекта изображена на рис. 6.13.

Подключая светодиодный индикатор к сдвиговому регистру, нужно соединить выводы A–G с выводами Q0–Q6 соответственно, а вывод DP соединить с выводом Q7.

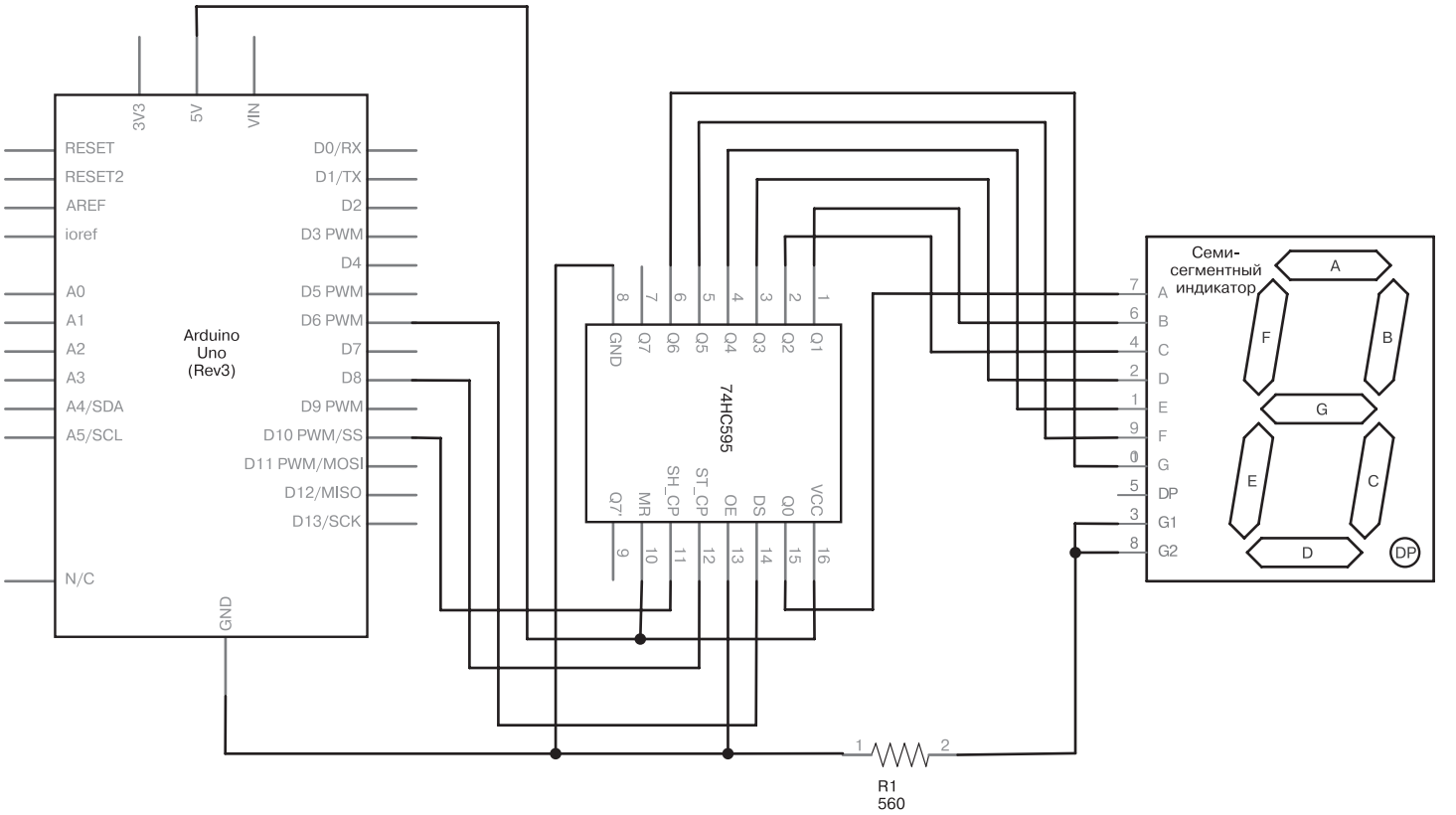


Рис. 6.13. Принципиальная схема[1] проекта 18

[1] Схема почти рабочая, но некорректная. Яркость сегментов будет зависеть от того, сколько сегментов включено. Должен быть индивидуальный резистор у каждого анода. Кроме того, автор забыл подключить анод DP. — *Примеч. науч. ред.*

Скетч

Скетч для проекта 18 хранит десятичные значения (см. табл. 6.2) в массиве `int digits[]`. Функция `void loop()` сначала последовательно посылает эти значения в сдвиговый регистр ❶, а после повторяет процесс с включенной десятичной точкой на индикаторе, добавляя 1 к значению, посылаемому в сдвиговый регистр ❷:

```
// Проект 18 — дисплей с одной цифрой

#define DATA   6 // К выводу 14 микросхемы 74HC595
#define LATCH  8 // К выводу 12 микросхемы 74HC595
#define CLOCK 10 // К выводу 11 микросхемы 74HC595

// Подготовить массив с комбинациями сегментов
// для цифр 0-9 и букв A-F (из табл. 6.2)
int digits[] = {252, 96, 218, 242, 102, 182, 190, 224,
                254, 246, 238, 62, 156, 122, 158, 142};

void setup()
{
  pinMode(LATCH, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(DATA, OUTPUT);
}

void loop()
{
  int i;
  for ( i = 0 ; i < 16 ; i++ ) // Вывести символы 0-9, A-F
  {
    digitalWrite(LATCH, LOW);
❶   shiftOut(DATA, CLOCK, LSBFIRST, digits[i]);
    digitalWrite(LATCH, HIGH);
    delay(250);
  }
  for ( i = 0 ; i < 16 ; i++ ) // Вывести символы 0-9, A-F с точкой
  {
    digitalWrite(LATCH, LOW);
❷   shiftOut(DATA, CLOCK, LSBFIRST, digits[i]+1); // +1, чтобы
                                                  // включить
                                                  // точку
    digitalWrite(LATCH, HIGH);
    delay(250);
  }
}
```

Рис. 6.14. Одна из цифр, отображаемых проектом 18

Семисегментные индикаторы светятся ярко и легко читаются. Например, на рис. 6.14 изображен индикатор со светящейся цифрой 9 и десятичной точкой.

Доработка скетча: отображение двух цифр

Чтобы использовать несколько сдвиговых регистров для управления дополнительными цифровыми выходами, соедините вывод 9 микросхемы 74HC595 (которая принимает данные с платы Arduino) с выводом 14 второго сдвигового регистра. После этого нужно послать два байта данных: первый для управления вторым сдвиговым регистром и второй — для управления первым:

```
digitalWrite(LATCH, LOW);
shiftOut(DATA, CLOCK, MSBFIRST, 254); // Данные для второй микросхемы 74HC595
shiftOut(DATA, CLOCK, MSBFIRST, 254); // Данные для первой микросхемы 74HC595
digitalWrite(LATCH, HIGH);
```

Проект 19: управление двумя семисегментными индикаторами

В этом проекте вы узнаете, как управлять двумя семисегментными светодиодными индикаторами и отображать двузначные числа.

Оборудование

Ниже перечислено необходимое оборудование:

- два сдвиговых регистра 74HC595;
- два семисегментных светодиодных индикатора с общим катодом;
- два резистора номиналом 560 Ом (R1–R2);
- одна большая макетная плата;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Схема

Схема для проекта с двумя индикаторами изображена на рис. 6.15.

Обратите внимание, что выводы «вход для тактовых импульсов» и «защелка» (11 и 12) сдвиговых регистров попарно соединены между собой и с платой Arduino. Линия передачи данных из Arduino (контакт 6) идет к выводу 14 первого сдвигового регистра и далее через вывод 9 первого регистра подключается к выводу 14 второго сдвигового регистра.

Для отображения чисел от 0 до 99 нужен скетч посложнее. Если число меньше 10, достаточно просто послать его и 0, чтобы на правом индикаторе отобразить цифру, а на левом — 0. Но если число больше или равно 10, необходимо определить составляющие его цифры и послать каждую в свой сдвиговый регистр. Для упрощения процедуры воспользуемся математической функцией деления по модулю.

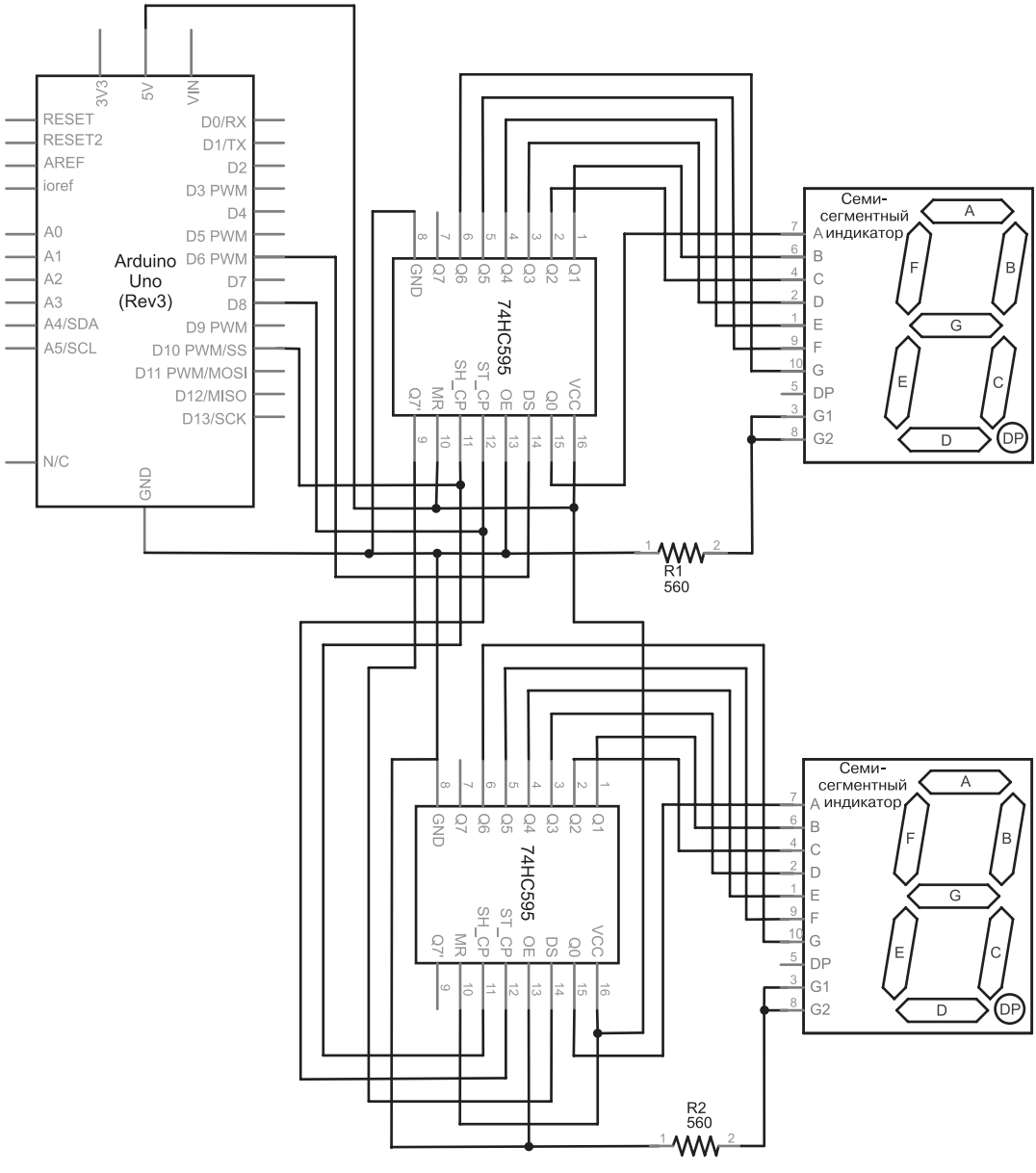


Рис. 6.15. Принципиальная схема[1] проекта 19

[1] Та же проблема, как в схеме выше. Яркость сегментов будет зависеть от того, сколько сегментов включено. — *Примеч. науч. ред.*

Деление по модулю

Деление по модулю — это функция, возвращающая остаток от деления. К примеру, деление по модулю 10 на 7 даст ответ 3. Иными словами, остаток от деления 10 на 7 равен 3. Операцию деления по модулю в языке C выполняет оператор `%`. Ниже показано, как он используется в скетчах:

```
int a = 8;
int b = 3;
c = a % b;
```

В этом примере переменная `c` получит значение 2. То есть для получения младшей цифры из двузначного числа следует использовать операцию деления по модулю, возвращающую остаток от деления двух чисел.

Для автоматизации отображения одно- и двузначных чисел мы напишем функцию `displayNumber()`. В ней будет использоваться операция деления по модулю для определения цифр в двузначных числах. К примеру, чтобы отобразить число 23, сначала нужно отделить левую цифру, разделив 23 на 10 — получится 2 (дробную часть игнорируем). Чтобы отделить правую цифру, разделите по модулю 23 на 10, что даст в результате 3.

```
// Проект 19 — управление двумя семисегментными индикаторами

#define DATA   6 // К выводу 14 микросхемы 74HC595
#define LATCH  8 // К выводу 12 микросхемы 74HC595
#define CLOCK 10 // К выводу 11 микросхемы 74HC595

// Подготовить массив с комбинациями сегментов
// для цифр 0–9 и букв A–F (из табл. 6.2)
int digits[] = {252, 96, 218, 242, 102, 182, 190, 224,
                254, 246, 238, 62, 156, 122, 158, 142};

void setup()
{
  pinMode(LATCH, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(DATA, OUTPUT);
}

void displayNumber(int n)
{
  int left, right=0;
❶ if (n < 10)
  {
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, LSBFIRST, digits[n]);
    shiftOut(DATA, CLOCK, LSBFIRST, 0);
```

```
      digitalWrite(LATCH, HIGH);
    }
    else if (n >= 10)
    {
❷     right = n % 10; // Остаток от деления числа на 10
      left = n / 10;  // Частное от деления числа на 10
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[right]);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[left]);
      digitalWrite(LATCH, HIGH);
    }
  }

❸ void loop()
  {
    int i;
    for ( i = 0 ; i < 100 ; i++ )
    {
      displayNumber(i);
      delay(100);
    }
  }
```

Функция `displayNumber()` сначала сравнивает отображаемое число с 10 ❶. Если число меньше, функция посылает в сдвиговые регистры его и «пустую» цифру — число 0. Но если число больше или равно 10, функция использует операции деления по модулю и деления ❷ для получения цифр и посылает их в сдвиговые регистры по отдельности. Наконец, в функции `void loop()` ❸ происходит вывод чисел от 0 до 99 с помощью `displayNumber()`.

Проект 20: цифровой термометр

В этом проекте мы добавим в схему вывода двузначных чисел температурный датчик TMP36 из главы 4 и создадим цифровой термометр. Алгоритм его работы прост: скетч будет читать выходное напряжение датчика TMP36 (используя прием из проекта 12) и преобразовывать его в градусы Цельсия.

Оборудование

Ниже перечислено необходимое оборудование:

- собранная схема с двумя сдвиговыми регистрами из проекта 19;
- один температурный датчик TMP36.

Подключите центральный вывод датчика TMP36 к контакту аналогового входа A5, левый — к контакту 5 V и правый — к контакту GND.

Скетч

Ниже приводится скетч для этого проекта:

```
// Проект 20 — цифровой термометр

#define DATA   6 // К выводу 14 микросхемы 74HC595
#define LATCH  8 // К выводу 12 микросхемы 74HC595
#define CLOCK 10 // К выводу 11 микросхемы 74HC595

int temp = 0;
float voltage = 0;
float celsius = 0;
float sensor = 0;
int digits[]={
  252, 96, 218, 242, 102, 182, 190, 224,
  254, 246, 238, 62, 156, 122, 158, 142
};

void setup()
{
  pinMode(LATCH, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(DATA, OUTPUT);
}

void displayNumber(int n)
{
  int left, right = 0;
  if (n < 10)
  {
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, LSBFIRST, digits[n]);
    shiftOut(DATA, CLOCK, LSBFIRST, digits[0]);
    digitalWrite(LATCH, HIGH);
  }
  if (n >= 10)
  {
    right = n % 10;
    left = n / 10;
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, LSBFIRST, digits[right]);
    shiftOut(DATA, CLOCK, LSBFIRST, digits[left]);
    digitalWrite(LATCH, HIGH);
  }
}

void loop()
{
  sensor = analogRead(5);
  voltage = (sensor * 5000) / 1024; // Преобразовать в милливольты
```

```
  voltage = voltage - 500;   // Учесть смещение
  celsius = voltage / 10;    // Преобразовать милливольты в градусы
  temp = int(celsius);       // Преобразовать вещественное значение
                             // температуры в целочисленное
  displayNumber(temp);
  delay(500);
}
```

Скетч очень прост и заимствует код из предыдущих проектов: `displayNumber()` из проекта 19 и вычисление температуры из проекта 12. Вызов `delay(500);` во второй с конца строке скетча препятствует слишком быстрой смене показаний, вызываемых колебаниями температуры.

Что дальше?

В этой главе вы овладели множеством базовых навыков, которые пригодятся в будущих проектах. Светодиодные индикаторы довольно надежны, поэтому не бойтесь экспериментировать с ними. Но их изобразительные возможности скудны, поэтому в следующей главе мы воспользуемся более мощными средствами для отображения текста и графики.

7

Расширение Arduino

В этой главе вы:

- познакомитесь с большим разнообразием плат расширения для Arduino;
- создадите собственную макетную плату расширения на основе платы ProtoShield;
- увидите, как расширить набор доступных функций с использованием библиотек для Arduino;
- научитесь пользоваться платой расширения с картой памяти для записи данных, которые затем можно проанализировать с помощью электронной таблицы;
- сконструируете устройство регистрации температуры;
- узнаете, как реализовать секундомер с использованием функций `micros()` и `millis()`;
- познакомитесь с прерываниями в Arduino и научитесь использовать их.

Другой способ расширения возможностей Arduino — использовать разнообразные платы расширения. *Плата расширения* — это печатная плата, подключаемая контактами к разъемам по краям платы Arduino. В первом проекте этой главы вы узнаете, как сделать свою плату расширения. Со временем вы научитесь создавать более долговечные конструкции, собирая их на *ProtoShield* — пустой макетной печатной плате для сборки собственной схемы.

Дальше вы познакомитесь с устройством для чтения/записи карт памяти. На его основе мы в этой главе создадим устройство регистрации температуры, записывающее замеры температуры на карту памяти в течение длительного времени.

Записанные с помощью платы расширения данные в дальнейшем можно перенести куда угодно для будущего анализа.

Вы познакомитесь с функциями `micros()` и `millis()` — очень удобными инструментами измерения интервалов времени, которые будут использованы в проекте реализации секундомера. В конце займемся исследованием прерываний.

Платы расширения

Расширить функциональные возможности платы Arduino можно с помощью дополнительных плат. В магазинах продаются сотни таких. Их можно комбинировать и соединять стопкой друг с другом. Один из популярных проектов, например, объединяет плату расширения GPS (определения географических координат) с платой расширения, содержащей карту памяти microSD. Итоговое устройство способно регистрировать и сохранять историю изменения координат с течением времени, к примеру путь, пройденный автомобилем или пешим путешественником. Из других проектов можно упомянуть сетевые адаптеры Ethernet, позволяющие подключить Arduino к интернету (рис. 7.1).

Рис. 7.1. Плата расширения с модулем Ethernet, подключенная к Arduino Uno

Приемники сигналов от спутников GPS позволяют определять местоположение Arduino (рис. 7.2). Интерфейсы с картами памяти microSD дают возможность сохранять данные на карте памяти (рис. 7.3).

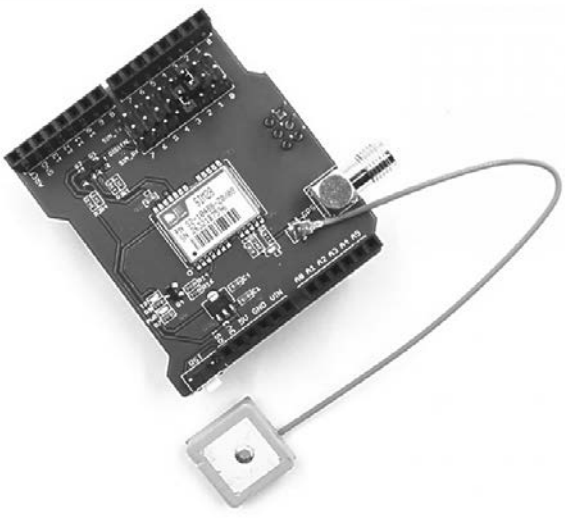


Рис. 7.2. Плата приемника GPS, подключенная к Arduino Uno

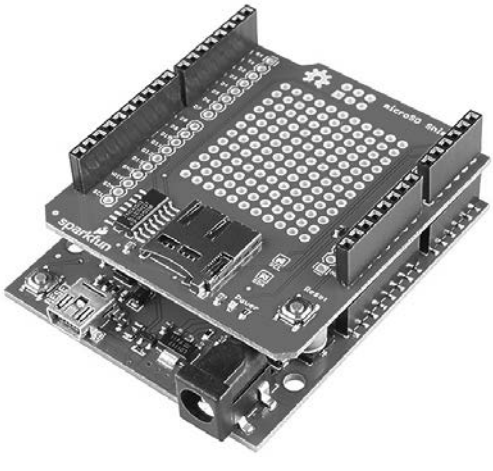


Рис. 7.3. Плата с держателем для карт памяти microSD

На рис. 7.4 показана плата Arduino Uno с подключенной платой расширения microSD для хранения данных, платой расширения Ethernet для доступа к интернету и платой расширения с жидкокристаллическим индикатором для отображения информации.

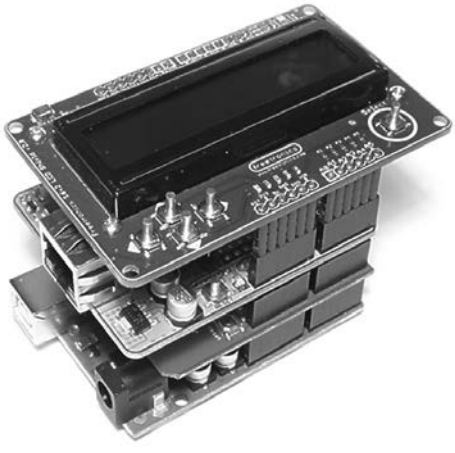


Рис. 7.4. Три платы расширения, подключенные к Arduino Uno «бутербродом»

ВНИМАНИЕ

При подключении нескольких плат расширения убедитесь, что они не используют одни и те же цифровые или аналоговые входы/выходы одновременно. Это может повредить всю конструкцию, поэтому будьте осторожны. Производители плат расширения обычно сопровождают свои изделия информацией об используемых ими входах/выходах.

Макетные платы ProtoShield

Платы расширения можно купить или сделать самим на основе макетной платы ProtoShield.

ProtoShield — это пустая печатная плата, которую можно использовать для изготовления своих плат расширения для Arduino. Она продается в собранном виде или в виде конструктора (рис. 7.5).

Кроме того, ProtoShield может быть отличной основой для макетирования без паяльника, так как позволяет размещать маленькие схемы в физических границах Arduino (рис. 7.6). С помощью клей-геля Blu-Tack или двусторонней клейкой ленты к печатной плате можно временно прикрепить небольшой модуль, умещающийся между рядами разъемов. Платы ProtoShield используются и в качестве основы для сборки постоянных схем, предварительно опробованных на макетной плате для сборки без пайки.

Конструирование устройств на плате ProtoShield требует предварительного планирования. Вы должны разработать принципиальную схему и схему размещения компонентов на плате. Готовое устройство можно спаять на плате, но перед этим

обязательно проверить его на макетной плате, чтобы убедиться, что оно работает. Для некоторых ProtoShield есть специальные файлы PDF со схемами, которые можно загрузить, напечатать и использовать для рисования схем к своим проектам.

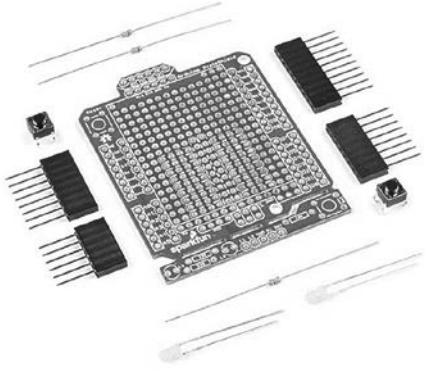


Рис. 7.5. Конструктор ProtoShield

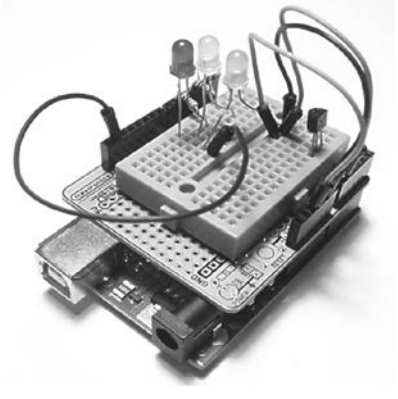


Рис. 7.6. Пример небольшого проекта, смонтированного на макетной печатной плате ProtoShield для сборки без пайки

Проект 21: создание собственной платы расширения

В этом проекте мы создадим свою плату расширения с двумя светодиодами и ограничивающими резисторами. Эта плата упростит работу со светодиодами, подключенными к цифровым выходам.

Оборудование

Ниже перечислено, что понадобится для этого проекта:

- одна пустая плата ProtoShield с впаянными контактами для подключения к Arduino;
- два светодиода любого цвета;
- два резистора номиналом 560 Ом;
- два резистора номиналом 10 кОм;
- две кнопки без фиксации;
- два конденсатора емкостью 100 нФ.

Схема

Принципиальная схема устройства изображена на рис. 7.7.

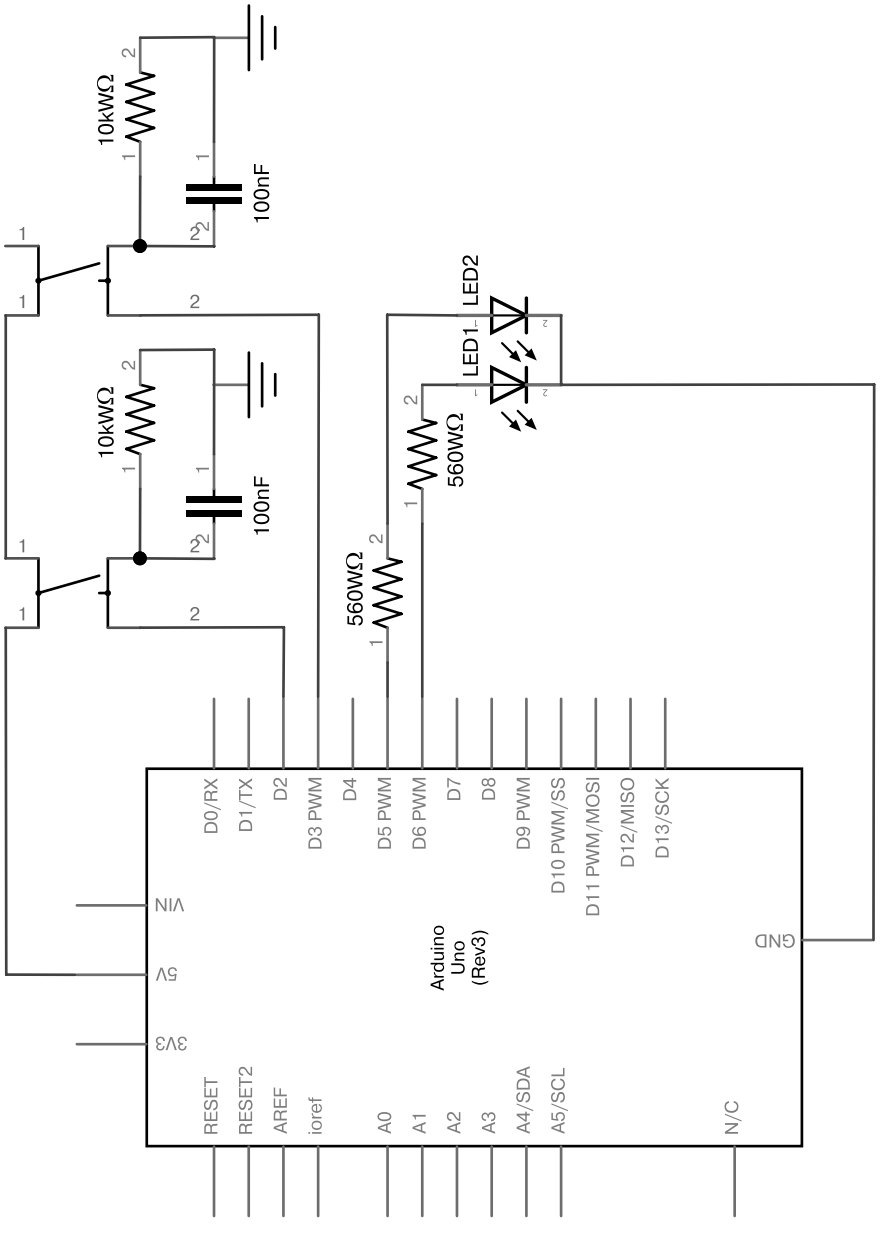


Рис. 7.7. Принципиальная схема для проекта 21

Топология макетной платы ProtoShield

Следующий наш шаг — знакомство с топологией отверстий на ProtoShield. Ряды и столбцы отверстий на этой плате должны совпадать с отверстиями на макетной плате для навесного монтажа. Но платы ProtoShield от разных производителей могут немного различаться, поэтому нужно выделить время, чтобы выяснить, как соединяются отверстия. На пустой плате ProtoShield (рис. 7.8) некоторые отверстия соединены токопроводящими дорожками, но многие остались изолированными. Это обеспечивает большую гибкость в использовании ProtoShield.

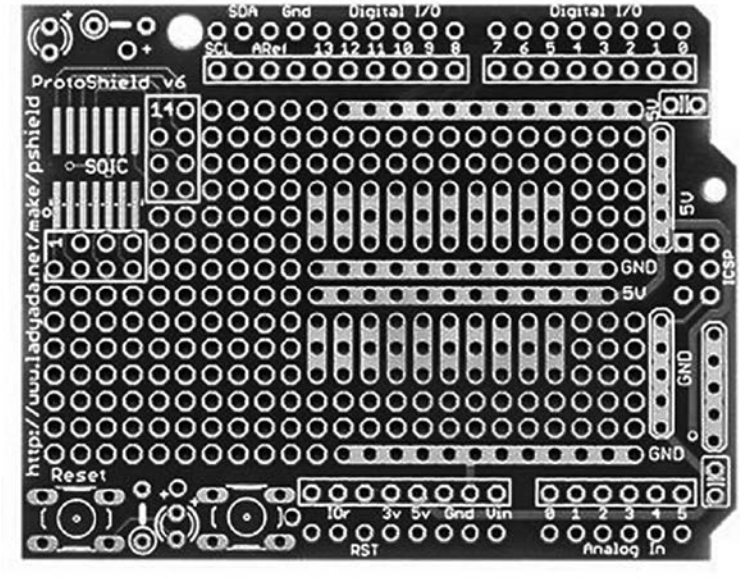


Рис. 7.8. Пустая плата Blank ProtoShield, вид сверху

Обратите внимание на две группы отверстий, окруженных прямоугольниками вдоль верхнего и нижнего краев ProtoShield: сюда должны впаиваться колодки с контактами для подключения к плате Arduino.

Проектирование

Вы должны преобразовать принципиальную схему на рис. 7.7 в схему размещения компонентов на плате ProtoShield. Лучше всего использовать для этого листы бумаги в клетку (рис. 7.9). Вы можете перенести на бумагу схему соединения отверстий на плате и поэкспериментировать, чтобы найти размещение компонентов, подходящее для вашей конкретной платы ProtoShield. Если нужной бумаги

у вас нет, напечатайте клетки на простой белой, воспользовавшись ссылкой http://www.printfreegraphpaper.com/.

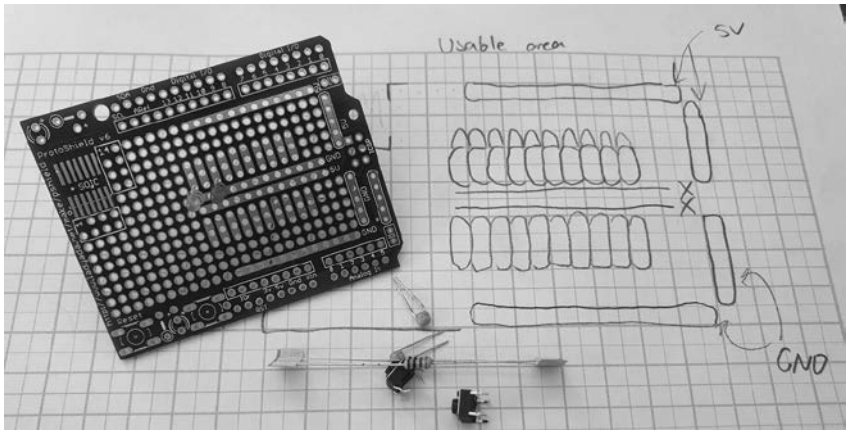


Рис. 7.9. Планирование будущей платы расширения

Когда план будет готов, проверьте, можно ли разместить фактические электронные компоненты на плате ProtoShield, как вы задумали, и не получилась ли компоновка чрезмерно плотной. Если на ProtoShield останется место для кнопки сброса, обязательно добавьте ее, потому что после подключения платы расширения к разъемам на плате Arduino доступ к стандартной кнопке сброса будет закрыт.

Пайка компонентов

После проверки расположения всех составных частей и работоспособности схемы приступайте к пайке компонентов. Пользоваться паяльником совсем не сложно. Дорогостоящие паяльные станции нам тоже не понадобятся — вполне достаточно простого паяльника мощностью 25–40 Вт (рис. 7.10).

ПРИМЕЧАНИЕ

Если вам не приходилось пользоваться паяльником, ознакомьтесь с иллюстрированной инструкцией: http://mightyohm.com/soldercomic/[1].

В процессе пайки компонентов иногда приходится соединять контакты небольшой каплей припоя и коротким отрезком провода (рис. 7.11).

[1] Перевод на русский язык можно найти по адресу https://geektimes.ru/post/256784/. — *Примеч. пер.*

Рис. 7.10. Паяльник

Рис. 7.11. Соединение контактов каплей припоя

По мере продвижения проверяйте каждый спаянный контакт, потому что ошибки проще найти и исправить *до* завершения проекта. Когда очередь дойдет до пайки четырех колодок с контактами, зафиксируйте их в разъемах другой платы расширения, чтобы избежать смещений (рис. 7.12).

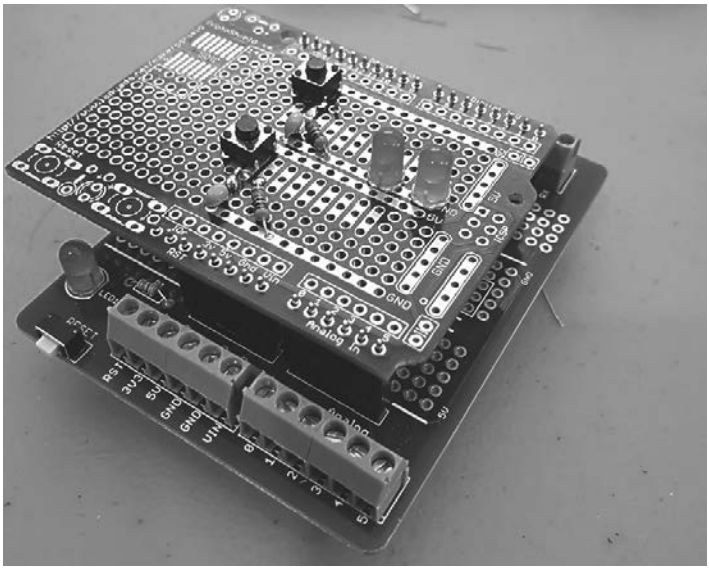


Рис. 7.12. Пайка колодок с контактами

На рис. 7.13 показано готовое устройство: наша плата расширения для Arduino с двумя светодиодами и двумя кнопками.

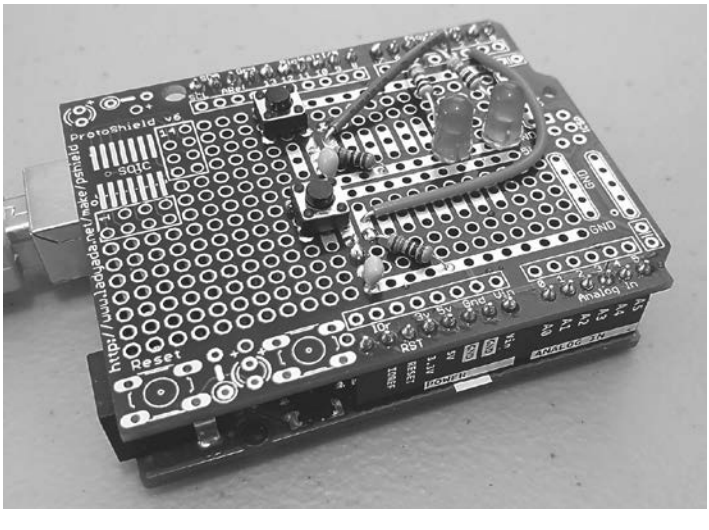


Рис. 7.13. Законченная плата расширения!

Проверка собранной платы ProtoShield

Прежде чем продолжить, хорошо бы протестировать кнопки и светодиоды на собранной плате ProtoShield. Скетч в листинге 7.1 использует две кнопки для включения и выключения светодиодов.

Листинг 7.1. Тестирование кнопок и светодиодов на плате ProtoShield

```
void setup()
{
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
}

void loop()
{
  if (digitalRead(2) == HIGH)
  {
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
  }
  if (digitalRead(3) == HIGH)
  {
    digitalWrite(5, LOW);
    digitalWrite(6, LOW);
  }
}
```

Расширение возможностей скетчей с помощью библиотек

Так же как платы расширения Arduino расширяют функциональные возможности оборудования, *библиотеки* могут добавлять полезные функции в наши скетчи. Эти функции могут давать возможность управлять дополнительным оборудованием на плате расширения. Любой может создать свою библиотеку, как это часто делают производители разных плат расширения Arduino для поддержки своего оборудования.

Среда разработки Arduino IDE уже содержит множество предустановленных библиотек. Чтобы подключить их к своим скетчам, достаточно выбрать в меню пункт Sketch ▸ Include Library (Скетч ▸ Подключить библиотеку), и вы увидите список предустановленных библиотек с такими именами, как Ethernet, LiquidCrystal, Servo и др. Если в каком-нибудь из последующих проектов понадобится библиотека, она будет описана более подробно.

После приобретения новой платы расширения обычно нужно загрузить и установить библиотеку для ее поддержки с сайта производителя или по указанной ссылке. Давайте рассмотрим оба метода, выполнив весь процесс установки библиотеки, необходимой для использования платы расширения microSD (см. рис. 7.3).

Загрузка библиотеки в виде ZIP-файла

Сначала загрузим и установим библиотеку в формате ZIP. Эта дополнительная библиотека нужна для чтения и записи данных на карты microSD и SD.

1. Откройте страницу https://github.com/greiman/SdFat/ и нажмите кнопку Code. Убедитесь, что выбрана вкладка HTPPS, и нажмите на Download ZIP (рис. 7.14).

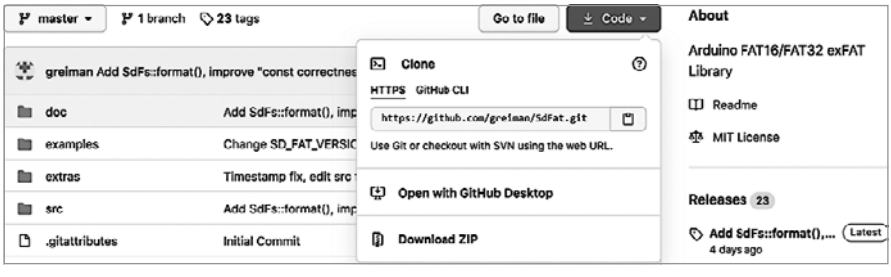


Рис. 7.14. Страница загрузки библиотеки

2. Через несколько секунд в папке Downloads (Загрузки) появится файл SdFat-master.zip, как показано на рис. 7.15. Если вы используете компьютер Apple, то ZIP-файл может быть распакован автоматически.

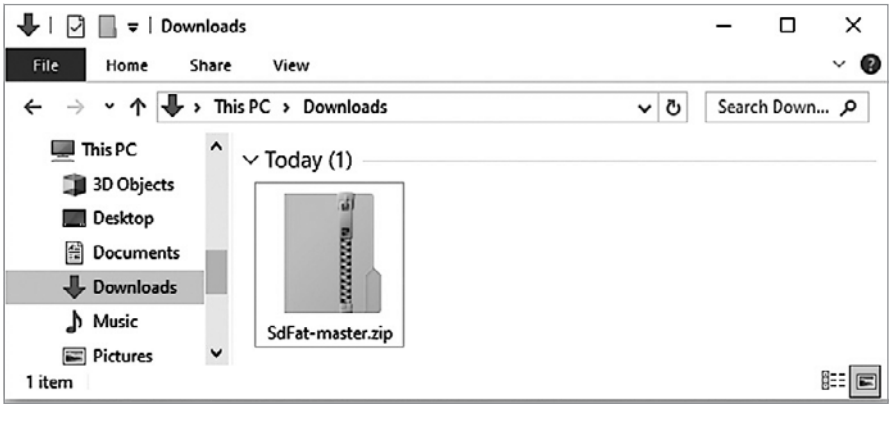


Рис. 7.15. Папка с загруженной библиотекой

3. Откройте Arduino IDE и выберите пункт меню Sketch ▸ Include Library ▸ Add .ZIP Library (Скетч ▸ Подключить библиотеку ▸ Добавить ZIP-библиотеку), как показано на рис. 7.16.

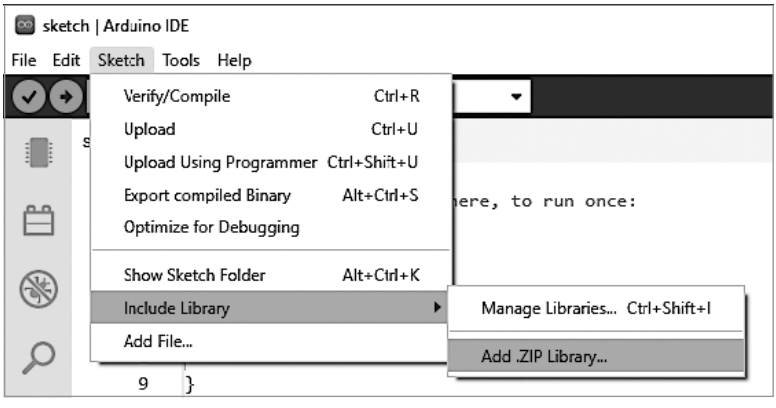


Рис. 7.16. Запуск процесса установки библиотеки

После появления диалога, как на рис. 7.17, перейдите в папку Downloads (Загрузки) или куда вы сохранили ZIP-файл, выберите файл SdFat-master.zip и нажмите Open (Открыть).

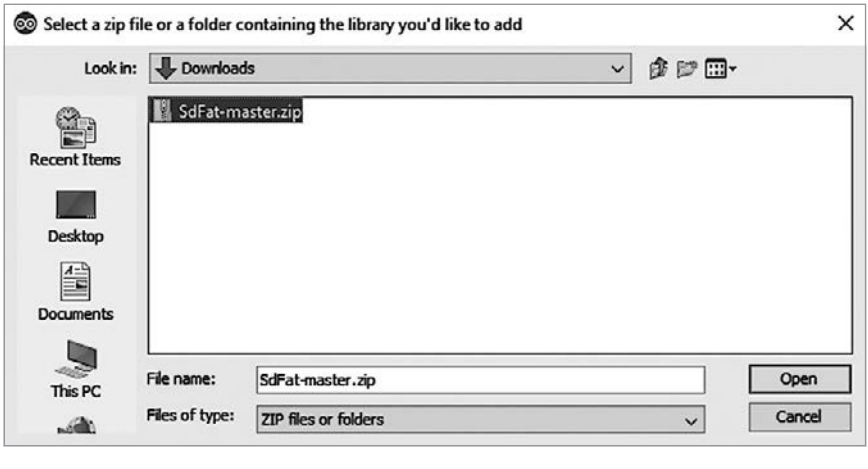


Рис. 7.17. Выбор ZIP-файла библиотеки

После этого Arduino IDE позаботится об установке библиотеки. Через несколько секунд вы увидите уведомление в окне вывода IDE (рис. 7.18).

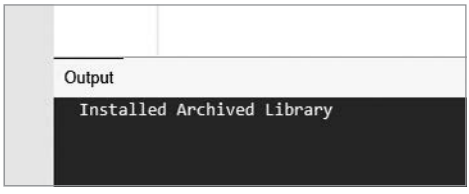


Рис. 7.18. Библиотека установлена благополучно

4. Проверьте корректность установки библиотеки SdFat и ее доступность для менеджера библиотек в IDE. Для этого щелкните на значке Library Manager (Менеджер библиотек) на панели слева в окне IDE. В верхнем поле поиска попробуйте найти библиотеку или просто прокрутите список вниз, пока не увидите свою. На рис. 7.19 показано, что библиотека SdFat есть в списке менеджера библиотек.

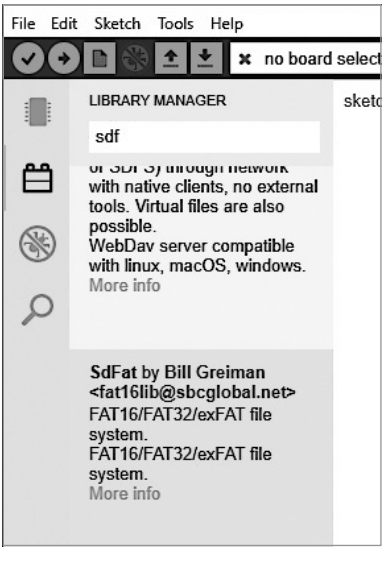


Рис. 7.19. Библиотека SdFat установлена и доступна для менеджера библиотек

Импортирование библиотеки Arduino с помощью менеджера библиотек

Другой способ установки библиотеки Arduino — с помощью встроенного в Arduino IDE менеджера библиотек Library Manager. Этот инструмент открывает доступ к онлайн-хранилищу библиотек для широкого использования, одобренных командой Arduino или просто очень популярных. Обычно к менеджеру библиотек обращаются по указанию производителей плат расширения.

Для примера давайте загрузим библиотеку FastLED, которая используется для управления популярным типом светодиодов RGB.

Откройте Arduino IDE, если вы еще этого не сделали, после чего откройте окно менеджера библиотек. Введите FastLED в поле поиска вверху (рис. 7.20). По мере ввода список библиотек будет обновляться в соответствии с введенной строкой, и в какой-то момент вы увидите то, что нужно вам.

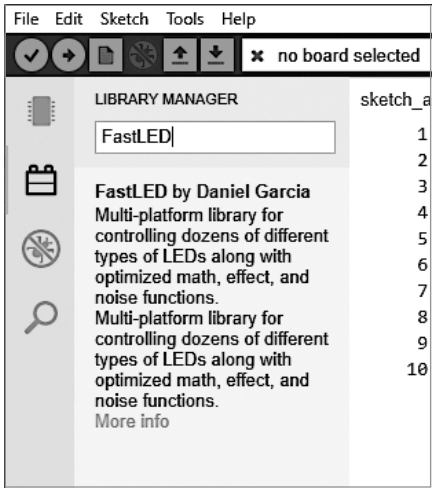


Рис. 7.20. Поиск библиотеки в онлайн-хранилище

Отыскав библиотеку в менеджере библиотек, наведите указатель мыши на ее описание — возможно, будет предложено выбрать версию библиотеки. Обычно отображается последняя версия, поэтому можно просто нажать Install (Установить) и дождаться завершения установки. Информация о ходе установки отображается в отдельном окне (рис. 7.21).

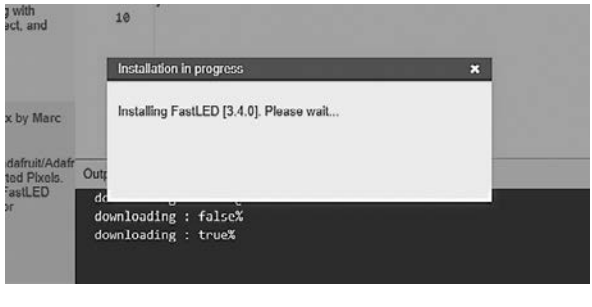


Рис. 7.21. Процесс установки библиотеки

При желании можете проверить корректность установки библиотеки, как было описано выше в этой главе.

Карты памяти microSD

Используя карты памяти SD или microSD с платой Arduino, можно организовать сохранение данных, поступающих из множества источников, таких как температурный датчик TMP36 из главы 4. На карте памяти можно хранить и данные для веб-сервера или любые другие файлы проекта. Для записи и хранения данных можно использовать карты памяти вроде тех, что изображены на рис. 7.22.

Рис. 7.22. Карта microSD емкостью 16 ГБ

С Arduino можно использовать карты памяти SD и microSD.

ПРИМЕЧАНИЕ

Прежде чем использовать новую карту памяти, ее нужно отформатировать. Для этого подключите ее к компьютеру и следуйте инструкциям вашей операционной системы по форматированию карт памяти.

Подключение модуля для чтения карт памяти

Для использования карты памяти нужно соединить шестью проводами модуль чтения карт с платой Arduino. Модули чтения карт обоих типов (microSD и SD) будут иметь одинаковые наборы контактов, которые должны быть подписаны (рис. 7.23).

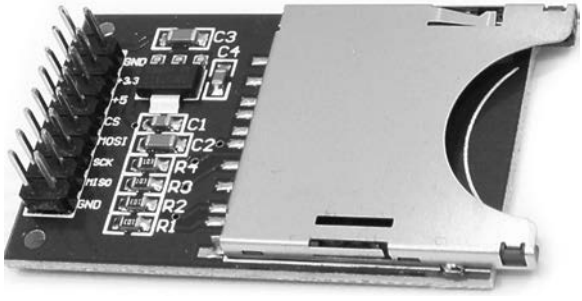


Рис. 7.23. Модуль для чтения карт SD

Соедините модуль с платой Arduino, руководствуясь табл. 7.1.

Таблица 7.1. Соединения контактов модуля чтения карт и Arduino

Подпись контакта модуля чтения карт	Контакт Arduino	Назначение контакта модуля чтения карт
5 V или Vcc	5 V	Питание
GND	GND	«Земля»
CS	D10	Выбор
MOSI	D11	Передача данных в Arduino
MISO	D12	Передача данных из Arduino
SCK	D13	Тактовые импульсы

Тестирование карты microSD

Отформатировав карту и подключив модуль чтения карт к Arduino, проверьте его работоспособность. Для этого выполните следующие шаги.

1. Вставьте карту в разъем модуля и подключите Arduino к компьютеру кабелем USB.
2. Запустите IDE и выберите пункт меню File ▸ Examples ▸ SdFat ▸ SdInfo (Файл ▸ Примеры ▸ SdFat ▸ SdInfo). После этого в IDE загрузится скетч примера.

3. Прокрутите скетч вниз до строки 36 и измените значение в объявлении `const int chipSelect` с `4` на `10` (рис. 7.24). Это необходимо, потому что номер используемого контакта зависит от конкретного оборудования. Потом загрузите скетч в Arduino.

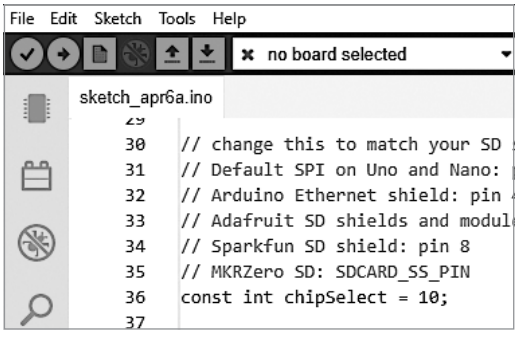


Рис. 7.24. Редактирование тестового скетча

4. Откройте окно Serial Monitor (Монитор порта), установите скорость обмена 9600 бод, нажмите любую клавишу на клавиатуре, а потом Enter. Спустя мгновение в области вывода появится информация о карте microSD (рис. 7.25).

```
Initializing SD card...Wiring is correct and a card is present.

Card type:         SDHC
Clusters:          490192
Blocks x Cluster:  64
Total Blocks:      31372288

Volume type is:    FAT32
Volume size (Kb):  15686144
Volume size (Mb):  15318
Volume size (Gb):  14.96

Files found on the card (name, date and size in bytes):
SYSTEM~1/     2019-09-24 16:52:30
  INDEXE~1      2019-09-24 16:52:30 76
```

Рис. 7.25. Результаты успешного тестирования карты microSD

Если в мониторе порта ничего не появилось, попробуйте выполнить следующие действия:

- отключите кабель USB от платы Arduino, извлеките и вновь вставьте карту microSD в разъем;

- руководствуясь табл. 7.1, проверьте правильность соединений;
- проверьте, установлена ли скорость 9600 бод в окне монитора порта, и убедитесь, что используемая плата Arduino совместима с обычной Arduino Uno. У некоторых моделей (например, Mega) иное расположение выводов интерфейса SPI, через который осуществляется обмен данными с платами расширения;
- отформатируйте карту microSD еще раз;
- если выполненные выше действия не помогли, попробуйте использовать другую карту памяти.

Перед подключением или отключением SD/microSD карты убедитесь, что ваша система отключена от USB и/или внешнего источника питания.

Проект 22: запись данных на карту памяти

В этом проекте мы используем карту памяти для сохранения данных, в частности — таблицы умножения.

Скетч

Для записи данных на карту памяти подключите плату расширения, вставьте карту microSD в держатель, введите и загрузите следующий скетч:

```
// Проект 29 – запись данных на карту памяти
#include <SD.h>
int b = 0;

void setup()
{
  Serial.begin(9600);
  Serial.print("Initializing SD card...");
  pinMode(10, OUTPUT);
  // Проверить наличие и работоспособность карты SD
  if (!SD.begin(10))
  {
    Serial.println("Card failed, or not present");
    // Остановить скетч
    return;
  }
  Serial.println("microSD card is ready");
}
void loop()
{
```

```
❶   // Создать файл для записи
    File dataFile = SD.open("DATA.TXT", FILE_WRITE);
    // Если файл готов, записать в него данные:
    if (dataFile)
❷   {
      for ( int a = 0 ; a < 11 ; a++ )
      {
        dataFile.print(a);
        dataFile.print(" multiplied by two is ");
        b = a * 2;
❸       dataFile.println(b, DEC);
      }
❹     dataFile.close(); // Закрыть файл, как только система
                        // завершит запись (обязательно)
    }
    // Если файл не готов, сообщить об ошибке:
    else
    {
      Serial.println("error opening DATA.TXT");
    }
    Serial.println("finished");
    do {} while (1);
  }
```

Скетч создаст на карте памяти microSD текстовый файл с именем `DATA.TXT`. Его содержимое можно увидеть на рис. 7.26.

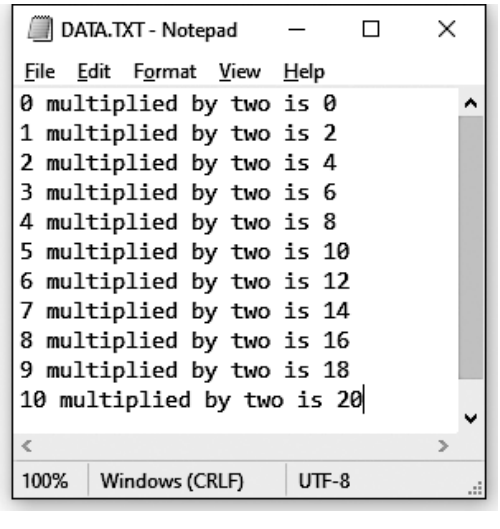


Рис. 7.26. Результаты работы проекта 22

Исследуем функцию `void loop()` в скетче и посмотрим, как она создает текстовый файл. Код в функции `void loop()` между ❶ и ❷ создает файл и открывает его для записи. Запись в текстовый файл осуществляется вызовом `dataFile.print()` или `dataFile.println()`.

Эти функции можно использовать по аналогии, например, с `Serial.println()`. Запись в файл осуществляется точно так же, как запись в монитор порта. В ❶ определяется имя создаваемого файла, не длиннее восьми символов, за которым должны следовать точка и три символа расширения, как в имени `DATA.TXT`.

В ❸ передается второй параметр `DEC`. Он сообщает, что переменная `b` хранит десятичное число, которое должно быть записано в текстовый файл. При записи вещественной переменной (типа `float`) можно указать число десятичных знаков для записи (не более шести).

По завершении записи данных в файл ❹ вызывается `dataFile.close()`, чтобы его закрыть. Если этот шаг опустить, компьютер не сможет прочитать созданный текстовый файл.

Проект 23: устройство регистрации температуры

Теперь, когда вы знаете, как записывать данные, попробуем измерять температуру каждую минуту в течение восьми часов и записывать результаты на карту microSD. Для этого объединим функции записи на карту microSD из проекта 22 со схемой измерения температуры из главы 4.

Оборудование

Ниже перечислено, что понадобится для этого проекта:

- один температурный датчик TMP36;
- одна макетная плата;
- несколько отрезков провода разной длины;
- плата расширения с картой памяти microSD;
- плата Arduino и кабель USB.

Вставьте карту microSD в держатель на плате расширения и подключите плату расширения к плате Arduino. Соедините левый (`5 V`) вывод датчика TMP36 с контактом `5 V` на Arduino, средний вывод — с аналоговым входом и правый вывод — с контактом `GND`.

Скетч

Введите и загрузите следующий скетч:

```
// Проект 23 — устройство регистрации температуры

#include <SD.h>
float sensor, voltage, celsius;

void setup()
{
  Serial.begin(9600);
  Serial.print("Initializing SD card...");
  pinMode(10, OUTPUT);

  // Проверить наличие и работоспособность карты microSD
  if (!SD.begin(10))
  {
    Serial.println("Card failed, or not present");
    // Остановить скетч
    return;
  }
  Serial.println("microSD card is ready");
}

void loop()
{
  // Создать файл для записи
  File dataFile = SD.open("DATA.TXT", FILE_WRITE);
  // Если файл готов, записать в него данные:
  if (dataFile)
  {
    for ( int a = 0 ; a < 481 ; a++ ) // 480 минут, или 8 часов
    {
      sensor = analogRead(0);
      voltage = (sensor * 5000) / 1024; // Преобразовать в милливольты
      voltage = voltage - 500;
      celsius = voltage / 10;
      dataFile.print(" Log: ");
      dataFile.print(a, DEC);
      dataFile.print(" Temperature: ");
      dataFile.print(celsius, 2);
      dataFile.println(" degrees C");
      delay(599900); // Ждать примерно одну минуту
    }
    dataFile.close(); // Обязательно
    Serial.println("Finished!");
    do {} while (1);
  }
}
```

До завершения работы скетча потребуется чуть больше восьми часов, но вы можете изменить эту продолжительность, уменьшив значение в вызове `delay(599900)`.

После завершения скетча извлеките карту microSD, вставьте в компьютер и откройте файл в текстовом редакторе (рис. 7.27).

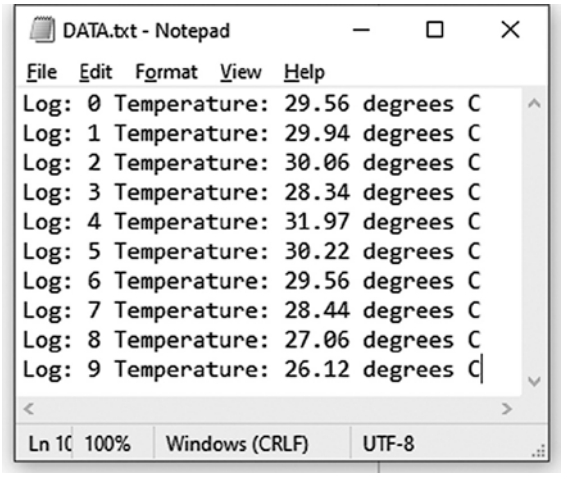


Рис. 7.27. Результаты работы проекта 23

Для более серьезного анализа сохраненных данных разделите текстовые строки, записываемые в файл, пробелами или двоеточиями, чтобы упростить импортирование файла в электронную таблицу. Файл можно переместить в OpenOffice Calc или Excel (рис. 7.28).

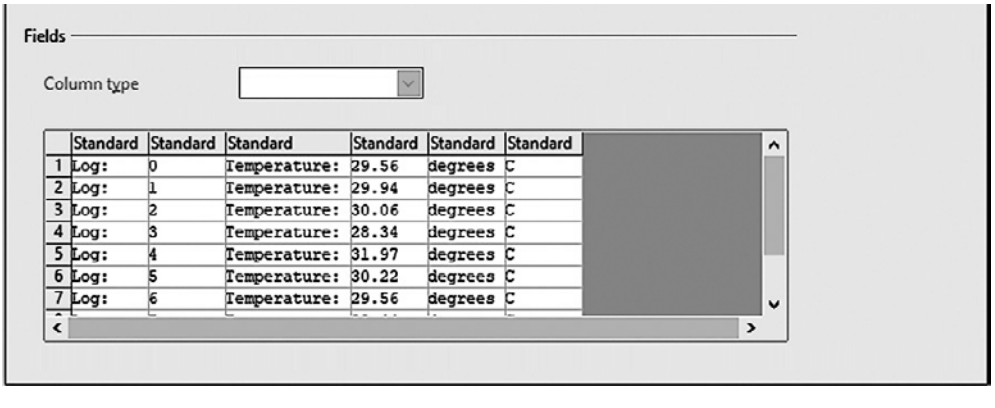
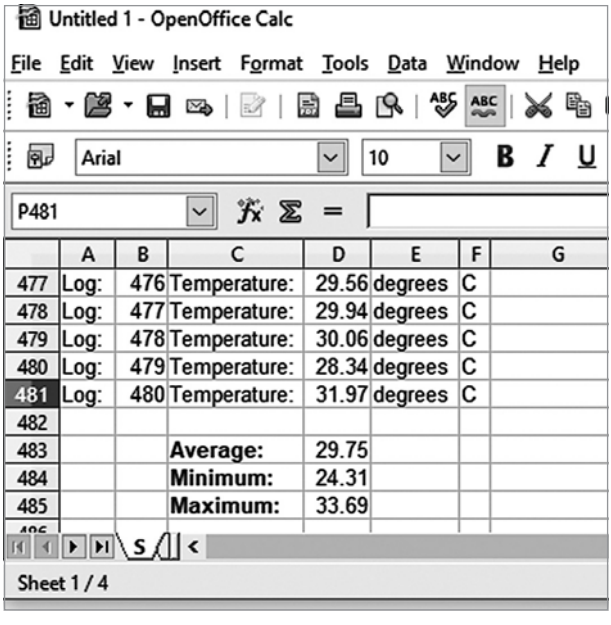


Рис. 7.28. Данные, импортированные в электронную таблицу

И затем проанализировать данные, как показано на рис. 7.29.



	A	B	C	D	E	F	G
477	Log:	476	Temperature:	29.56	degrees	C	
478	Log:	477	Temperature:	29.94	degrees	C	
479	Log:	478	Temperature:	30.06	degrees	C	
480	Log:	479	Temperature:	28.34	degrees	C	
481	Log:	480	Temperature:	31.97	degrees	C	
482							
483			**Average:**	29.75			
484			**Minimum:**	24.31			
485			**Maximum:**	33.69			

Рис. 7.29. Анализ результатов измерения температуры

Образцы замеров температуры можно приспособить для собственных проектов анализа данных. Рассмотренные принципы применимы для записи любых данных, которые только способна сгенерировать система Arduino.

Хронометраж с применением millis() и micros()

Каждый раз после запуска скетча Arduino начинает фиксировать ход времени в миллисекундах и микросекундах. Миллисекунда — это одна тысячная (0,001) секунды, а микросекунда — одна миллионная (0,000 001). Эти значения можно использовать в скетчах для измерения интервалов времени.

Следующие функции возвращают значения времени, хранящиеся в переменных типа `unsigned long`:

```
unsigned long a,b;
a = micros();
b = millis();
```

Из-за ограничений типа `unsigned long` значение сбрасывается в 0 по достижении 4 294 967 295. Это примерно 50 дней для `millis()` и 70 минут для `micros()`.

Кроме того, из-за ограничений микропроцессора на плате Arduino, `micros()` всегда возвращает значения, кратные четырем.

Воспользуемся этими функциями и определим, сколько времени нужно плате Arduino, чтобы перевести цифровой выход из состояния `LOW` в `HIGH` и обратно. Для этого прочитаем результат `micros()` до и после вызова функции `digitalWrite()`, определим разность и выведем ее на монитор порта. Нам потребуются только плата Arduino и кабель USB.

Введите и загрузите скетч из листинга 7.2.

Листинг 7.2. Хронометраж изменения состояния цифрового выхода с помощью micros()

```
unsigned long start, finished, elapsed;

void setup()
{
  Serial.begin(9600);
  pinMode(3, OUTPUT);
  digitalWrite(3, LOW);
}

void loop()
{
❶  start = micros();
   digitalWrite(3, HIGH);
❷  finished = micros();
❸  elapsed = finished – start;
   Serial.print("LOW to HIGH: ");
   Serial.print(elapsed);
   Serial.println(" microseconds");
   delay(1000);

❹  start = micros();
   digitalWrite(3, LOW);
   finished = micros();
   elapsed = finished - start;
   Serial.print("HIGH to LOW: ");
   Serial.print(elapsed);
   Serial.println(" microseconds");
   delay(1000);
}
```

Скетч получает отметки времени с помощью `micros()` до и после вызова функции `digitalWrite(HIGH)` (❶ и ❷), вычисляет разность и выводит ее в монитор порта ❸. Затем операции повторяются для измерения времени переключения цифрового выхода в прежнее состояние ❹.

Теперь откройте окно монитора порта, чтобы увидеть результаты (рис. 7.30).

```
HIGH to LOW 8 microseconds
LOW to HIGH: 8 microseconds
HIGH to LOW 8 microseconds
LOW to HIGH: 8 microseconds
HIGH to LOW 8 microseconds
LOW to HIGH: 8 microseconds
HIGH to LOW 8 microseconds
LOW to HIGH: 8 microseconds
HIGH to LOW 8 microseconds
LOW to HIGH: 8 microseconds
HIGH to LOW 8 microseconds
LOW to HIGH: 8 microseconds
HIGH to LOW 8 microseconds
```

Рис. 7.30. Результаты работы скетча из листинга 7.2

Точность измерения составляет 4 микросекунды, полученная продолжительность — 8 микросекунд. Это значит, что реальная продолжительность больше 4 микросекунд и меньше или равна 8.

Проект 24: секундомер

Теперь, когда мы научились измерять интервалы времени между двумя событиями, сконструируем на основе Arduino простой секундомер. В нем будут использоваться две кнопки: одна для запуска или сброса отсчета, другая — для остановки подсчета и отображения прошедшего времени. Скетч будет постоянно проверять состояние двух кнопок. В момент нажатия на кнопку запуска он сохранит значение `millis()`, а при нажатии кнопки останова — сохранит новое значение `millis()`. Наша собственная функция `displayResult()` преобразует измеренный интервал времени из миллисекунд в часы, минуты и секунды. В итоге преобразованное время мы увидим в окне монитора порта.

Оборудование

Используйте плату ProtoShield и добавьте следующие элементы:

- одну макетную плату;
- две кнопки без фиксации (S1 и S2);
- два резистора номиналом 10 кОм (R1 and R2);
- несколько отрезков провода разной длины;
- плату Arduino и кабель USB.

Схема

Принципиальная схема для проекта изображена на рис. 7.31.

ПРИМЕЧАНИЕ

Эта схема понадобится нам в следующем проекте, поэтому не разбирайте ее по завершении экспериментов!

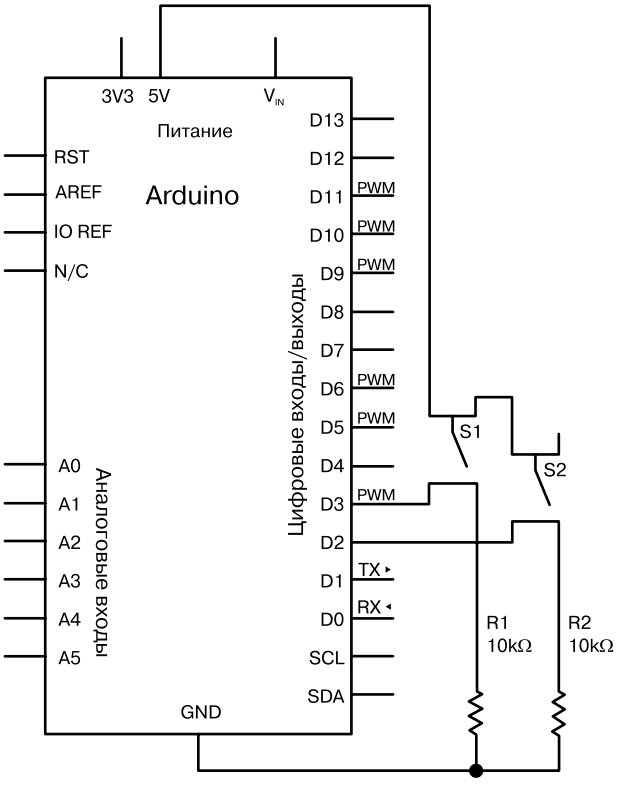


Рис. 7.31. Принципиальная схема для проекта 24

Скетч

Введите и загрузите следующий скетч:

```
// Проект 24 – секундомер

unsigned long start, finished, elapsed;
```

```
void setup()
{
  Serial.begin(9600);
❶ pinMode(2, INPUT); // Кнопка запуска
  pinMode(3, INPUT); // Кнопка останова
  Serial.println("Press 1 for Start/reset, 2 for elapsed time");
}

void displayResult()
{
  float h, m, s, ms;
  unsigned long over;

❷ elapsed = finished - start;

  h    = int(elapsed / 3600000);
  over = elapsed % 3600000;
  m    = int(over / 60000);
  over = over % 60000;
  s    = int(over / 1000);
  ms   = over % 1000;

  Serial.print("Raw elapsed time: ");
  Serial.println(elapsed);
  Serial.print("Elapsed time: ");
  Serial.print(h, 0);
  Serial.print("h ");
  Serial.print(m, 0);
  Serial.print("m ");
  Serial.print(s, 0);
  Serial.print("s ");
  Serial.print(ms, 0);
  Serial.println("ms");
  Serial.println();
}

void loop()
{
❸ if (digitalRead(2) == HIGH)
  {
    start = millis();
    delay(200); // Защита от дребезга контактов
    Serial.println("Started...");
  }
❹ if (digitalRead(3) == HIGH)
  {
    finished = millis();
    delay(200); // Защита от дребезга контактов
    displayResult();
  }
}
```

Наш секундомер сделать несложно. В строке ❶ выполняется настройка выводов на работу в режиме цифровых входов, к которым подключены кнопки запуска и останова. В ❸, если нажата кнопка запуска, Arduino запоминает возвращаемое функцией `millis()` значение, чтобы использовать его в вычислениях после нажатия кнопки останова ❹. После нажатия кнопки останова вызывается функция `displayResult()`, которая выведет вычисленный интервал времени ❷ в монитор порта.

На рис. 7.32 показаны результаты в окне монитора порта.

```
Started...
Press 1 for Start/reset, 2 for elapsed time
Raw elapsed time: 4368
Elapsed time: 0h 0m 4s 368ms

Raw elapsed time: 13662
Elapsed time: 0h 0m 13s 662ms

Raw elapsed time: 392366
Elapsed time: 0h 6m 32s 366ms

Raw elapsed time: 5052371
Elapsed time: 1h 24m 12s 371ms

Raw elapsed time: 5077826
Elapsed time: 1h 24m 37s 826ms
```

Рис. 7.32. Вывод проекта 24

Прерывания

Прерывание в мире Arduino — это сигнал, позволяющий вызвать функцию в скетче в любое время. Например, когда изменяется состояние цифрового входа или когда зафиксировано событие таймера. Прерывания отлично подходят для приостановки нормальной работы скетча и вызова функции (например, обрабатывающей факт нажатия кнопки). Такие функции часто называют *обработчиками прерываний*.

Во время прерывания нормальная работа программы временно приостанавливается, вызывается функция обработки прерывания, и по завершении этой функции работа программы возобновляется с того места, где была прервана.

Имейте в виду, что функции обработки прерываний должны быть максимально короткими и простыми. Их назначение — быстро выполнять свою работу. Если функция обработки прерываний выполняет такую же операцию, как в главном цикле программы, выполнение этой операции в главном цикле временно приостанавливается.

К примеру, если главный цикл регулярно посылает строку `Hello` в последовательный порт, а функция обработки прерываний посылает последовательность символов `---`, то в окне монитора последовательного порта можно увидеть любой из следующих вариантов вывода: `H---ello`, `He---llo`, `Hel---lo`, `Hell---o` или `Hello---`.

Плата Arduino Uno поддерживает два прерывания, связанные с цифровыми выводами 2 и 3. При правильной настройке аппаратура Arduino следит за уровнем напряжения, приложенного к выводам. Когда напряжение изменяется определенным способом (например, после нажатия кнопки), генерируется прерывание, вызывающее соответствующую функцию, которая выводит текст Stop Pressing Me!.

Режимы прерываний

Есть четыре вида изменений *(режимов)*, вызывающих прерывания:

- `LOW` — к цифровому входу приложен низкий уровень напряжения;
- `CHANGE` — уровень напряжения на цифровом входе изменился с высокого на низкий или наоборот;
- `RISING` — уровень напряжения на цифровом входе изменился с низкого на высокий;
- `FALLING` — уровень напряжения на цифровом входе изменился с высокого на низкий.

Чтобы определить факт нажатия кнопки, подключенной к контакту с поддержкой прерывания, используется режим `RISING`. Или, например, если вокруг вашего сада протянут тонкий провод (соединяющий контакт 5 V с контактом цифрового входа), можно использовать режим `FALLING` для определения момента, когда кто-то порвет провод.

ПРИМЕЧАНИЕ

Функции delay() и Serial.available() не будут работать в функциях, вызываемых по прерываниям.

Настройка прерываний

Для настройки прерываний добавьте следующие инструкции в функцию `void setup()`:

```
attachInterrupt(0, function, mode);
attachInterrupt(1, function, mode);
```

Здесь 0 соответствует цифровому контакту 2, 1 — контакту 3, *`function`* — имя функции, вызываемой по прерываниям, и *`mode`* — один из четырех режимов прерываний.

Включение и выключение прерываний

Иногда в скетче нужно отключить прерывания. Можно отключить одно конкретное:

```
detachInterrupt(digitalPinToInterrupt(pin))
```

где *pin* — номер используемого цифрового входа. Или можно отключить все прерывания:

```
noInterrupts(); // выключить прерывания
```

А затем вновь включить:

```
interrupts(); // включить прерывания
```

Прерывания чувствительны и работают очень быстро, поэтому они очень полезны при выполнении срочных операций, например в момент нажатия кнопки «экстренная остановка».

Проект 25: использование прерываний

Для демонстрации прерываний вернемся к схеме из проекта 24. Этот пример включает и выключает встроенный светодиод каждые 500 миллисекунд, одновременно проверяя оба контакта с поддержкой прерываний. В момент нажатия кнопки, связанной с прерыванием 0, в монитор порта выводится значение `micros()`, а в момент нажатия кнопки, связанной с прерыванием 1, — `millis()`.

Скетч

Введите и загрузите следующий скетч:

```
// Проект 25 — использование прерываний

#define LED 13

void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  attachInterrupt(0, displayMicros, RISING);
  attachInterrupt(1, displayMillis, RISING);
}

❶ void displayMicros()
{
  Serial.write("micros() = ");
  Serial.println(micros());
}
```

```
❷ void displayMillis()
  {
    Serial.write("millis() = ");
    Serial.println(millis());
  }

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

Скетч включает и выключает встроенный светодиод на плате, как можно видеть в функции `void loop()` ❸. Когда возникает прерывание 0, вызывается функция `displayMicros()` ❶, а во время прерывания 1 — функция `displayMillis()` ❷. По завершении любой из функций скетч продолжает выполнение `void loop()`.

Откройте окно монитора порта и понажимайте обе кнопки. В результате в области вывода должны появиться значения `millis()` и `micros()` (рис. 7.33).

```
millis() = 3769
micros() = 4555052
millis() = 5483
micros() = 6328740
millis() = 7155
micros() = 7673876
millis() = 8339
micros() = 8874420
millis() = 9511
micros() = 10024460
millis() = 10618
micros() = 11339048
millis() = 11542
micros() = 12413384
millis() = 12880
micros() = 13292308
millis() = 13809
micros() = 14309052
```

Рис. 7.33. Вывод проекта 25

Что дальше?

В этой главе вы познакомились с несколькими инструментами и возможностями для создания и улучшения своих проектов. Дальше мы будем применять другие платы расширения для Arduino, измерять интервалы времени с помощью прерываний и сохранять результаты на карте памяти.

8

Светодиодные цифровые табло и матрицы

В этой главе вы:

- познакомитесь со светодиодными цифровыми табло на основе MAX7219;
- создадите свой цифровой секундомер;
- познакомитесь с модулями светодиодных матриц на основе MAX7219;
- создадите светодиодное текстовое табло с бегущей строкой.

Может, светодиодные цифровые табло (как в цифровых будильниках, например) представляют и не самые передовые технологии отображения, но они легко читаются и, что важно, легко комбинируются с платами Arduino.

В главе 6 вы узнали, как использовать одно- и двузначные цифровые светодиодные индикаторы. Но одновременное отображение множества цифр может быть затруднительным — потребуется больше проводов, управляющих микросхем и т. д., которые нужно смонтировать на плате. К счастью, есть популярная микросхема, способная контролировать максимум 64 светодиодов (восьмиразрядное цифровое табло) всего по трем проводам — светодиодный драйвер MAX7219 компании Maxim Integrated.

MAX7219 выпускается в корпусах двух типов: для выводного монтажа (то есть имеет металлические ножки, которые можно вставлять в отверстия в печатной или макетной плате без пайки, как показано на рис. 8.1) и для поверхностного монтажа (рис. 8.2).

Рис. 8.1. Микросхема MAX7219 в корпусе для выводного монтажа

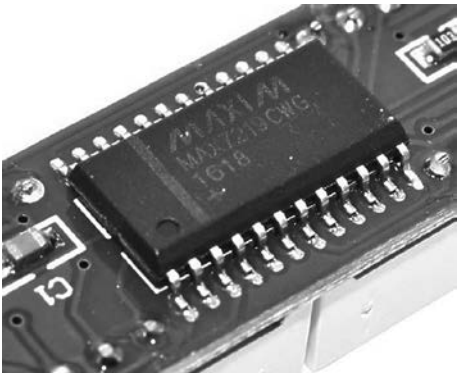


Рис. 8.2. Микросхема MAX7219 в корпусе для поверхностного монтажа

В главе вы узнаете, как использовать микросхему MAX7219 для управления светодиодными цифровыми табло, насчитывающими до восьми цифр. После вы познакомитесь с приемами использования MAX7219 для управления модулями светодиодных матриц, из которых можно составлять текстовые табло с бегущей строкой.

Светодиодные цифровые табло

Светодиодные цифровые табло на основе MAX7219 бывают разных форм и размеров, обычно имеют от четырех до восьми цифр. В наших примерах мы используем табло с восемью цифрами. Оно широкодоступно и имеет отличное соотношение «цена/качество» (рис. 8.3).

Рис. 8.3. Восьмиразрядное светодиодное цифровое табло

Эти табло оснащены микросхемой MAX7219 в корпусе для поверхностного монтажа (рис. 8.2). Ее можно увидеть на обратной стороне. Обычно на табло предусмотрено

место для колодки с контактами, к которым подключаются провода управления. Припаяйте такую к табло, как показано на рис. 8.4, если вы этого еще не сделали.

Рис. 8.4. Колодка с контактами на восьмиразрядном цифровом табло

Прежде чем использовать цифровое табло, нужно соединить его с платой Arduino пятью проводами. Это легко сделать с помощью проводов со штекерами и гнездами, которые нужно подключить к колодке контактами, ранее припаянными к табло. Выполните подключения, руководствуясь табл. 8.1.

Таблица 8.1. Карта соединений цифрового табло с Arduino

Контакт табло	Контакт Arduino	Назначение контакта табло
Vcc	5 V	Питание (+)
GND	GND	Питание (–), или «земля»
DIN	D12	Входные данные
CS	D10	Выбор микросхемы
CLK	D11	Тактовые импульсы

Установка библиотеки

Для MAX7219 в Arduino есть несколько библиотек. Они отличаются поддерживаемыми конфигурациями цифровых табло. Дальше мы будем использовать библиотеку LedControl. Загрузите ZIP-файл с библиотекой, доступный по адресу https://github.com/wayoda/LedControl/. Нажмите Code и в открывшемся меню выберите Download ZIP (рис. 8.5).

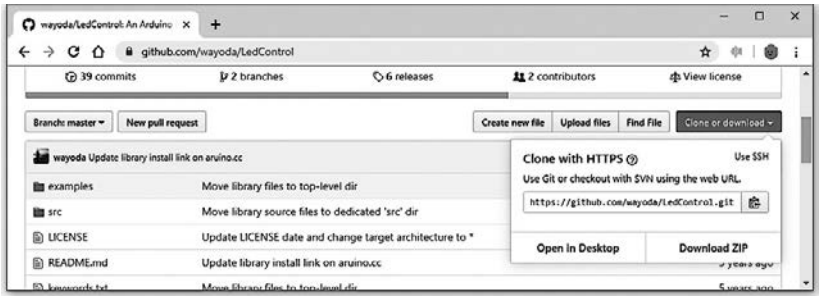


Рис. 8.5. Страница загрузки библиотеки LedControl

Установите загруженную библиотеку, как описано в главе 7. Теперь, чтобы поближе познакомиться с цифровым табло, мы рассмотрим демонстрационный скетч со всеми необходимыми функциями. Введите в IDE и загрузите скетч из листинга 8.1.

Листинг 8.1. Скетч для демонстрации управления цифровым табло

```
❶#include "LedControl.h" // Требуемая библиотека
LedControl lc = LedControl(12, 11, 10, 1);

void setup()
{
❷ lc.shutdown(0, false); // Включить табло
  lc.setIntensity(0, 3); // Настроить яркость
  lc.clearDisplay(0);    // Очистить индикаторы
}

void loop()
{
  // Числа с десятичной точкой
  for (int a = 0; a < 8; a++)
  {
❸   lc.setDigit(0, a, a, true);
    delay(500);
    lc.clearDisplay(0) ; // Очистить индикаторы
  }
  // Дефисы
  for (int a = 0; a < 8; a++)
  {
❹   lc.setChar(0, a, '-', false);
    delay(500);
    lc.clearDisplay(0) ; // Очистить индикаторы
  }

  // Числа без десятичной точки
  for (int a = 0; a < 8; a++)
  {
    lc.setDigit(0, a, a, false);
```

```
    delay(500);
    lc.clearDisplay(0) ; // Очистить индикаторы
  }
❺ // Вывести "abcdef"
  lc.setDigit(0, 5, 10, false);
  lc.setDigit(0, 4, 11, false);
  lc.setDigit(0, 3, 12, false);
  lc.setDigit(0, 2, 13, false);
  lc.setDigit(0, 1, 14, false);
  lc.setDigit(0, 0, 15, false);
  delay(500);
  lc.clearDisplay(0) ; // Очистить индикаторы
}
```

Разберем работу этого скетча. В строке ❶ подключается библиотека функций для управления табло. У функции `LedControl()` четыре параметра:

```
LedControl lc = LedControl(12, 11, 10, 1);
```

Первые три определяют контакты цифровых входов/выходов, к которым подключено табло (см. табл. 8.1), а четвертый — это количество табло, подключенных к Arduino, в данном случае — одно (можно последовательно включить несколько табло).

В ❷ вызываются три функции настройки аспектов отображения. Первая включает или выключает табло:

```
lc.shutdown(0, false);
```

Первый параметр определяет порядковый номер табло. Здесь передается 0, потому что к Arduino подключено только одно табло. Если подключить несколько, второе будет иметь порядковый номер 1, третье — 2 и т. д.

Второй параметр определяет, включить или выключить табло: `false` означает «включить», а `true` — «выключить».

Вторая функция настраивает яркость свечения светодиодов на табло:

```
lc.setIntensity(0, 3);
```

Первый параметр определяет порядковый номер табло. Второй — уровень яркости, который может принимать значения от 0 до 15 включительно.

Третья функция просто гасит все светодиоды:

```
lc.clearDisplay(0);
```

Это отличный способ стереть прежде отображавшиеся данные.

В строке ❸ вызывается функция `setDigit()` для вывода цифры на табло:

```
lc.setDigit(0, a, b, true);
```

Первый параметр определяет порядковый номер табло. Второй — позицию цифры на табло. Для 8-разрядного табло это значение может изменяться от 7 (крайняя левая цифра) до 0 (крайняя правая цифра). Третий параметр задает цифру для отображения (от 0 до 9). Передав в третьем параметре числа от 10 до 15, можно вывести буквы от A до F, как показывает фрагмент кода в ❺. И четвертый параметр управляет десятичной точкой: значение `true` включает ее, а `false` — выключает.

Еще с помощью функции `setChar()` можно вывести символы A–F, H, L, P, дефис, точку и подчеркивание вызовом, как показано в ❹:

```
lc.setChar(0, a, '-', false);
```

Она принимает те же параметры, что и `setDigit()`, за исключением третьего, где должен передаваться символ в одинарных кавычках.

Мы познакомились со всеми командами отображения чисел и символов на табло. Теперь применим их на практике.

Проект 26: цифровой секундомер

В проекте 24 (см. главу 7) вы познакомились с приемами измерения интервалов времени, а выше в этой главе вы узнали, как использовать цифровое табло. Теперь вы можете объединить все вместе для создания цифрового секундомера. Этот проект не дотягивает до олимпийских стандартов, но все еще очень интересен. Секундомер сможет отображать миллисекунды, секунды, минуты и до девяти часов.

Для реализации проекта подключите к Arduino макетную плату ProtoShield (или аналог) из главы 7 и цифровое табло, представленное выше в этой главе. После просто загрузите в Arduino следующий скетч:

```
// Проект 26 — цифровой секундомер

#include "LedControl.h" // Требуемая библиотека

LedControl lc = LedControl(12, 11, 10, 1);
unsigned long starting, finished, elapsed;

void setup()
{
  pinMode(2, INPUT);     // Кнопка запуска
  pinMode(3, INPUT);     // Кнопка останова
  lc.shutdown(0, false); // Включить табло
  lc.setIntensity(0, 3); // Настроить яркость
  lc.clearDisplay(0);    // Очистить индикаторы
  starting = millis();
}
```

```
❶ void displayResultLED()
{
  float h, m, s, ms;
  int m1, m2, s1, s2, ms1, ms2, ms3;
  unsigned long over;
  finished = millis();
  elapsed = finished - starting;

❷ h = int(elapsed / 3600000);
  over = elapsed % 3600000;
  m = int(over / 60000);
  over = over % 60000;
  s = int(over / 1000);
  ms = over % 1000;

❸ // Вывести часы
lc.setDigit(0, 7, h, true);

  // Вывести минуты
  m1 = m / 10;
  m2 = int(m) % 10;
  lc.setDigit(0, 6, m1, false);
  lc.setDigit(0, 5, m2, true);

  // Вывести секунды
  s1 = s / 10;
  s2 = int(s) % 10;
  lc.setDigit(0, 4, s1, false);
  lc.setDigit(0, 3, s2, true);

  // Вывести миллисекунды
  ms1 = int(ms / 100);
  ms2 = (int((ms / 10)) % 10);
  ms3 = int(ms) % 10;
  lc.setDigit(0, 2, ms1, false);
  lc.setDigit(0, 1, ms2, false);
  lc.setDigit(0, 0, ms2, false);
}

void loop()
{
❹ if (digitalRead(2) == HIGH) // Начать отсчет
  {
    starting = millis();
    delay(200); // Защита от дребезга контактов
  }
❺ if (digitalRead(3) == HIGH) // Приостановить изменение показаний на табло на 5 с
  {
    finished = millis();
    delay(5000); // Защита от дребезга контактов
  }
  displayResultLED();
}
```

Через мгновение после загрузки скетча на табло начнется отсчет (рис. 8.6).

Рис. 8.6. Цифровой секундомер в работе

Так же как и в проекте 24, для измерения времени в этом скетче используется функция `millis()`. Вычисление и отображение времени мы поместили в функцию `void displayResultLED()` ❶.

В ❷ можно видеть, как измеренное время в миллисекундах разбивается на часы, минуты, секунды и миллисекунды. Затем цифры на табло заполняются слева направо соответствующими значениями, начиная с часов ❸. Управляется секундомер просто: когда нажимается кнопка, подключенная к цифровому входу 2, счетчик сбрасывается в ноль за счет установки начального момента времени (`starting`) равным текущему значению, возвращаемому `millis()` ❹. После нажатия кнопки, связанной с цифровым входом 3, изменение цифр на табло приостанавливается. Эта функция идеально подходит для чтения промежуточных результатов. Но помните, что отсчет времени при этом продолжается и изменение показаний на табло возобновится примерно через пять секунд.

Этот проект можно легко приспособить для отображения данных в более простом формате (часов, минут и секунд) или для измерения более длительных интервалов времени (до 24 часов). А пока перейдем к более сложному проекту с использованием светодиодных матриц.

Проект 27: использование модулей светодиодных матриц

Микросхема MAX7219 может контролировать максимум 64 светодиодов. В предыдущем проекте мы отображали это цифрами. В этом мы используем модули со светодиодами, размещенными в виде матрицы 8 × 8. Они идеально подходят для создания более интересных устройств (табло для отображения фиксированного текста или бегущей строки).

Светодиодные матрицы обычно продаются по отдельности или наборами по четыре штуки (рис. 8.7).

Можно встретить и комплекты для самостоятельной сборки матриц. Но выгода будет минимальной, поэтому лучше сэкономить время и купить уже собранные

матрицы. Светодиодные матрицы вставляются в разъемы на модуле, как показано на рис. 8.8. Это позволяет легко менять цвета.

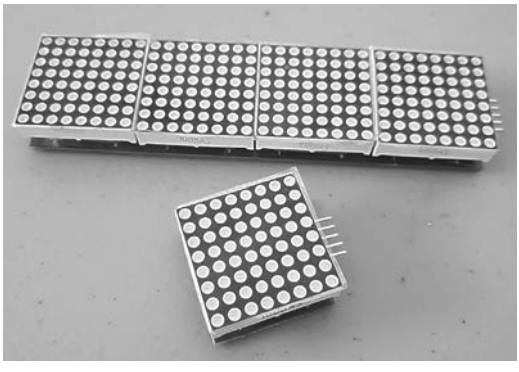


Рис. 8.7. Модули со светодиодными матрицами

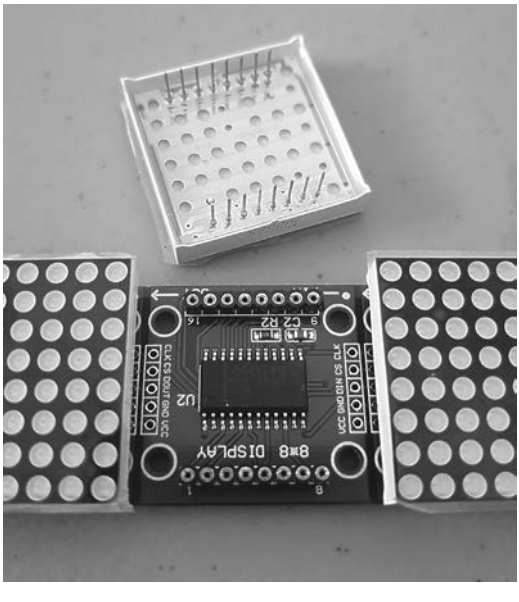


Рис. 8.8. Съемные светодиодные матрицы

Будьте осторожны с установкой светодиодных матриц в модули. Некоторые из них имеют легко гнущиеся контакты. В большинстве случаев к модулям матриц еще нужно припаять колодки с контактами. Обычно они входят в комплект поставки модуля и аккуратно вставляются в предусмотренные отверстия на плате (рис. 8.9).

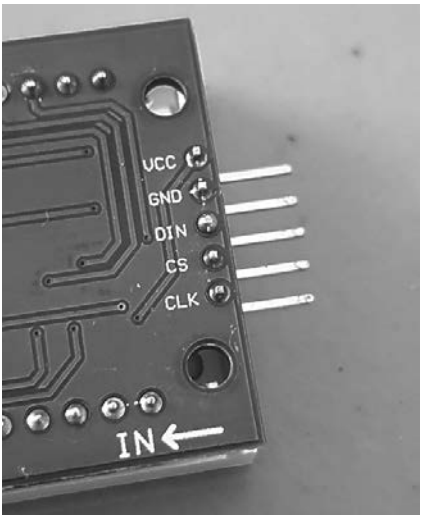


Рис. 8.9. Колодка с контактами, припаянная к модулю матрицы

И снова, прежде чем использовать модули со светодиодными матрицами, нужно соединить их с платой Arduino пятью проводами, как мы делали это с цифровым табло. Выполните подключения, руководствуясь табл. 8.2.

Таблица 8.2. Карта соединений модуля светодиодной матрицы с Arduino

Контакт модуля	Контакт Arduino	Назначение контакта модуля
Vcc	5 V	Питание (+)
GND	GND	Питание (–), или «земля»
DIN	D11	Входные данные
CS	D9	Выбор
CLK	D13	Тактовые импульсы

Установка библиотеки

Для работы с этими модулями нужна еще одна библиотека. Откройте страницу https://github.com/bartoszbielawski/LEDMatrixDriver/. Нажмите кнопку Code и в меню выберите Download ZIP (рис. 8.10).

Затем установите загруженную библиотеку, как описано в главе 7. Введите и загрузите следующий скетч (напоминаю, что весь код, который приводится в книге, можно загрузить на странице https://nostarch.com/arduino-workshop-2nd-edition/).

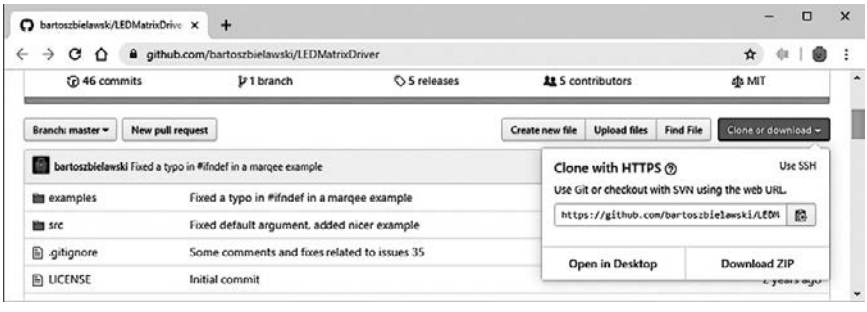


Рис. 8.10. Страница загрузки библиотеки LEDMatrixDriver

ПРИМЕЧАНИЕ

Если вы используете только один модуль со светодиодной матрицей, то измените значение в const int LEDMATRIX_SEGMENTS = 4 в строке 7 с 4 на 1.

```
// Проект 27 — использование модулей светодиодных матриц
❶ #include <LEDMatrixDriver.hpp>

const uint8_t LEDMATRIX_CS_PIN = 9;

// Количество подключенных модулей светодиодных матриц
const int LEDMATRIX_SEGMENTS = 4;
const int LEDMATRIX_WIDTH = LEDMATRIX_SEGMENTS * 8;
LEDMatrixDriver lmd(LEDMATRIX_SEGMENTS, LEDMATRIX_CS_PIN);

// Текст для отображения
❷ char text[] = "** LED MATRIX DEMO! ** (1234567890) ++
\"ABCDEFGHIJKLMNOPQRSTUVWXYZ\" ++ <$%/=?'.@,> --";

// Скорость прокрутки (чем меньше значение, тем выше скорость)
❸ const int ANIM_DELAY = 30;

void setup() {
❹ // Инициализация табло
  lmd.setEnabled(true);
  lmd.setIntensity(2); // 0 = low, 10 = high
}

int x = 0, y = 0; // Начинать с левого верхнего угла

// Определение шрифта
❺ byte font[95][8] = { {0, 0, 0, 0, 0, 0, 0, 0},    // ПРОБЕЛ
  {0x10, 0x18, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18}, // !
  {0x28, 0x28, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00}, // "
  {0x00, 0x0a, 0x7f, 0x14, 0x28, 0xfe, 0x50, 0x00}, // #
  {0x10, 0x38, 0x54, 0x70, 0x1c, 0x54, 0x38, 0x10}, // $
  {0x00, 0x60, 0x66, 0x08, 0x10, 0x66, 0x06, 0x00}, // %
```

```
{0, 0, 0, 0, 0, 0, 0, 0}, // &
{0x00, 0x10, 0x18, 0x18, 0x08, 0x00, 0x00, 0x00}, // '
{0x02, 0x04, 0x08, 0x08, 0x08, 0x08, 0x08, 0x04}, // (
{0x40, 0x20, 0x10, 0x10, 0x10, 0x10, 0x10, 0x20}, // )
{0x00, 0x10, 0x54, 0x38, 0x10, 0x38, 0x54, 0x10}, // *
{0x00, 0x08, 0x08, 0x08, 0x7f, 0x08, 0x08, 0x08}, // +
{0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x08}, // ,
{0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00}, // -
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x06}, // .
{0x00, 0x04, 0x04, 0x08, 0x10, 0x20, 0x40, 0x40}, // /
{0x00, 0x38, 0x44, 0x4c, 0x54, 0x64, 0x44, 0x38}, // 0
{0x04, 0x0c, 0x14, 0x24, 0x04, 0x04, 0x04, 0x04}, // 1
{0x00, 0x30, 0x48, 0x04, 0x04, 0x38, 0x40, 0x7c}, // 2
{0x00, 0x38, 0x04, 0x04, 0x18, 0x04, 0x44, 0x38}, // 3
{0x00, 0x04, 0x0c, 0x14, 0x24, 0x7e, 0x04, 0x04}, // 4
{0x00, 0x7c, 0x40, 0x40, 0x78, 0x04, 0x04, 0x38}, // 5
{0x00, 0x38, 0x40, 0x40, 0x78, 0x44, 0x44, 0x38}, // 6
{0x00, 0x7c, 0x04, 0x04, 0x08, 0x08, 0x10, 0x10}, // 7
{0x00, 0x3c, 0x44, 0x44, 0x38, 0x44, 0x44, 0x78}, // 8
{0x00, 0x38, 0x44, 0x44, 0x3c, 0x04, 0x04, 0x78}, // 9
{0x00, 0x18, 0x18, 0x00, 0x00, 0x18, 0x18, 0x00}, // :
{0x00, 0x18, 0x18, 0x00, 0x00, 0x18, 0x18, 0x08}, // ;
{0x00, 0x10, 0x20, 0x40, 0x80, 0x40, 0x20, 0x10}, // <
{0x00, 0x00, 0x7e, 0x00, 0x00, 0xfc, 0x00, 0x00}, // =
{0x00, 0x08, 0x04, 0x02, 0x01, 0x02, 0x04, 0x08}, // >
{0x00, 0x38, 0x44, 0x04, 0x08, 0x10, 0x00, 0x10}, // ?
{0x00, 0x30, 0x48, 0xba, 0xba, 0x84, 0x78, 0x00}, // @
{0x00, 0x1c, 0x22, 0x42, 0x42, 0x7e, 0x42, 0x42}, // A
{0x00, 0x78, 0x44, 0x44, 0x78, 0x44, 0x44, 0x7c}, // B
{0x00, 0x3c, 0x44, 0x40, 0x40, 0x40, 0x44, 0x7c}, // C
{0x00, 0x7c, 0x42, 0x42, 0x42, 0x42, 0x44, 0x78}, // D
{0x00, 0x78, 0x40, 0x40, 0x70, 0x40, 0x40, 0x7c}, // E
{0x00, 0x7c, 0x40, 0x40, 0x78, 0x40, 0x40, 0x40}, // F
{0x00, 0x3c, 0x40, 0x40, 0x5c, 0x44, 0x44, 0x78}, // G
{0x00, 0x42, 0x42, 0x42, 0x7e, 0x42, 0x42, 0x42}, // H
{0x00, 0x7c, 0x10, 0x10, 0x10, 0x10, 0x10, 0x7e}, // I
{0x00, 0x7e, 0x02, 0x02, 0x02, 0x02, 0x04, 0x38}, // J
{0x00, 0x44, 0x48, 0x50, 0x60, 0x50, 0x48, 0x44}, // K
{0x00, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x7c}, // L
{0x00, 0x82, 0xc6, 0xaa, 0x92, 0x82, 0x82, 0x82}, // M
{0x00, 0x42, 0x42, 0x62, 0x52, 0x4a, 0x46, 0x42}, // N
{0x00, 0x3c, 0x42, 0x42, 0x42, 0x42, 0x44, 0x38}, // O
{0x00, 0x78, 0x44, 0x44, 0x48, 0x70, 0x40, 0x40}, // P
{0x00, 0x3c, 0x42, 0x42, 0x52, 0x4a, 0x44, 0x3a}, // Q
{0x00, 0x78, 0x44, 0x44, 0x78, 0x50, 0x48, 0x44}, // R
{0x00, 0x38, 0x40, 0x40, 0x38, 0x04, 0x04, 0x78}, // S
{0x00, 0x7e, 0x90, 0x10, 0x10, 0x10, 0x10, 0x10}, // T
{0x00, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x3e}, // U
{0x00, 0x42, 0x42, 0x42, 0x42, 0x44, 0x28, 0x10}, // V
{0x80, 0x82, 0x82, 0x92, 0x92, 0x92, 0x94, 0x78}, // W
{0x00, 0x42, 0x42, 0x24, 0x18, 0x24, 0x42, 0x42}, // X
{0x00, 0x44, 0x44, 0x28, 0x10, 0x10, 0x10, 0x10}, // Y
```

```
  {0x00, 0x7c, 0x04, 0x08, 0x7c, 0x20, 0x40, 0xfe}, // Z
};

❻ void drawString(char* text, int len, int x, int y)
{
  for ( int idx = 0; idx < len; idx ++ )
  {
    int c = text[idx] - 32;

    // Прервать вывод, если произошел выход за границы отображаемой области
    if ( x + idx * 8 > LEDMATRIX_WIDTH )
      return;

    // Рисовать, только если символ видим
    if ( 8 + x + idx * 8 > 0 )
      drawSprite( font[c], x + idx * 8, y, 8, 8 );
  }
}

❼ void scrollText()
{
  int len = strlen(text);
  drawString(text, len, x, 0);
  lmd.display();

  delay(ANIM_DELAY);

  if ( --x < len * -8 ) {
    x = LEDMATRIX_WIDTH;
  }
}

void loop()
{
  scrollText();
}
```

Через несколько секунд после загрузки скетча на табло, собранном из модулей со светодиодными матрицами, должна появиться бегущая строка, прокручиваемая справа налево.

Теперь разберемся, как работает этот скетч. Он получился длинным, но не пугайтесь. В ❶ мы подключаем библиотеку с нужными функциями и настраиваем модули. В ❷ определятся массив символов с отображаемым на табло текстом, который позже при желании можно изменить. Вы можете и подкорректировать скорость прокрутки, изменив значение в ❸: чем меньше число, тем выше скорость прокрутки.

В ❹ вызываются две функции. Функция `setEnabled()` включает или выключает модуль:

```
lmd.setEnabled(true);
```

А функция `setIntensity()` настраивает яркость свечения светодиодов:

```
lmd.setIntensity(x);
```

Она принимает значение уровня яркости от 0 (минимальный) до 9 (максимальный).

В огромном массиве ❺ определяется шрифт для отображения символов. Мы рассмотрим его в следующем разделе. Наконец, для работы табло нужны функции `drawstring()` ❻ и `scrollText()` ❼.

Шрифт для отображения символов

Вы можете определить, как будут выглядеть символы на табло, изменив содержимое массива `byte font` ❺. Для начала напомню, что каждая матрица состоит из восьми рядов по восемь светодиодов в каждом. Значит, для изображения любого символа можно использовать 64 светодиода.

Каждый ряд светодиодов определяется шестнадцатеричным числом, а восемь таких чисел представляют изображение символа. Например, буква N определяется так:

```
{0x00, 0x42, 0x42, 0x62, 0x52, 0x4a, 0x46, 0x42}, // N
```

Чтобы посмотреть, как будет выглядеть символ, преобразуем шестнадцатеричные числа в двоичные. К примеру, наша N, преобразованная из шестнадцатеричного представления в двоичное, выглядит так:

```
0 0 0 0 0 0 0 0 = 0x00
0 1 0 0 0 0 1 0 = 0x42
0 1 0 0 0 0 1 0 = 0x42
0 1 1 0 0 0 1 0 = 0x62
0 1 0 1 0 0 1 0 = 0x52
0 1 0 0 1 0 1 0 = 0x4a
0 1 0 0 0 1 1 0 = 0x46
0 1 0 0 0 0 1 0 = 0x42
```

Здесь видно, как единицы образуют изображение символа на фоне нулей. Единицы — это включенные светодиоды, а нули — выключенные. Итак, чтобы создать свои символы, просто пройдите путь в обратном направлении. Например, симпатичный улыбающийся смайлик можно представить так:

```
0 1 1 1 1 1 1 0 = 0x7e
1 0 0 0 0 0 0 1 = 0x81
1 0 1 0 0 1 0 1 = 0xa5
1 0 0 0 0 0 0 1 = 0x81
1 0 1 0 0 1 0 1 = 0xa5
1 0 0 1 1 0 0 1 = 0x99
1 0 0 0 0 0 0 1 = 0x81
0 1 1 1 1 1 1 0 = 0x7e
```

В массиве он будет выглядеть следующим образом:

```
{0x7e,0x81,0xa5,0x81,0xa5,0x99,0x81,0x7e} // смайлик
```

Вы можете заменить существующую строку в массиве `font` новыми данными или добавить их в конец массива. Если вы решите добавить строку в конец, увеличьте первый параметр в объявлении `byte font`, чтобы он равнялся количеству символов, определяемых массивом (в данном случае 96):

```
byte font[96][8]
```

У вас, наверное, уже возник вопрос — как обозначить свой символ в скетче. Библиотека отображения использует нумерацию символов в таблице ASCII, которую можно найти по адресу https://www.arduino.cc/en/Reference/ASCIIchart/.

Если вы добавите еще один символ после последнего, определенного в скетче (по умолчанию Z), то следующим символом в таблице ASCII будет `[`. То есть, чтобы отобразить на табло строку из трех смайликов, нужно определить строку с отображаемым текстом так:

```
char text[] = "[ [ [ ";
```

Пример вывода этой строки на табло показан на рис. 8.11.

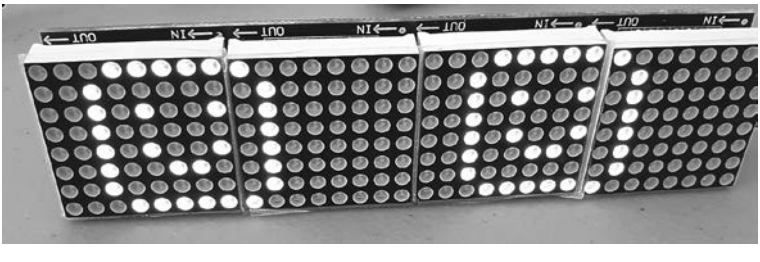


Рис. 8.11. Использование нестандартного символа для отображения смайликов

Что дальше?

Теперь, зная, как использовать светодиодные цифровые табло и матрицы, вы без труда сможете применять их в своих проектах. Но есть и другие устройства отображения информации. Переверните страницу, чтобы познакомиться с жидкокристаллическими индикаторами.

9

Жидкокристаллические индикаторы

В этой главе вы:

- научитесь пользоваться символьными жидкокристаллическими индикаторами для отображения текста и числовых данных;
- узнаете, как создавать свои символы для отображения на жидкокристаллических индикаторах;
- освоите отображение текста и данных на цветных жидкокристаллических индикаторах;
- создадите цифровой термометр с памятью, отображающий историю изменения температуры в виде графика.

В некоторых проектах нужно отображать информацию на чем-то помимо монитора настольного компьютера. Проще всего для этой цели использовать жидкокристаллические индикаторы (ЖКИ) вместе с платой Arduino. Текст, нестандартные символы и числовые данные можно отображать на символьных ЖКИ, а цветную графику — на графических.

Символьные жидкокристаллические индикаторы

Жидкокристаллические индикаторы, отображающие информацию в символах (текст и числа), — самые недорогие и простые в использовании среди всех ЖКИ. Они могут быть разных размеров, измеряемых числом и длиной отображаемых строк. У некоторых есть подсветка и возможность выбирать цвет символов

и фона. С Arduino можно использовать любые жидкокристаллические индикаторы с HD44780- или KS0066-совместимым интерфейсом и напряжением питания подсветки 5 В. Для начала попробуем использовать ЖКИ с подсветкой, способный отображать две строки по 16 символов в каждой (рис. 9.1).

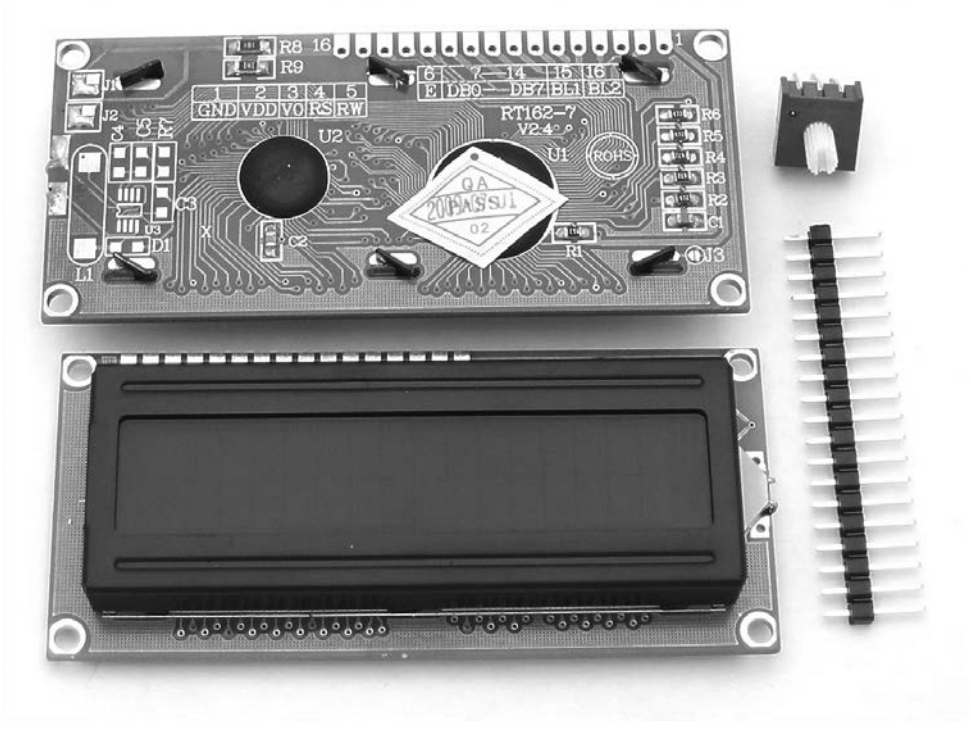


Рис. 9.1. Жидкокристаллический индикатор с подстроечным резистором и колодкой контактов

Подстроечный резистор (переменный резистор для ЖКИ) имеет сопротивление 10 кОм и используется для регулировки контрастности индикатора. Колодка контактов впаивается в ряд отверстий в верхней части платы с ЖКИ, чтобы можно было подключать индикатор прямо к макетной плате.

Отверстия вдоль верхнего края платы с ЖКИ пронумерованы от 1 до 16. Отверстие с номером 1, расположенное ближе к углу платы, на схеме на рис. 9.2 подписано как VSS и подключается к «земле». В редких случаях, когда ЖКИ имеет напряжение питания подсветки 4,2 В, нужно включить диод 1N4004 между контактом 5 V на плате Arduino и контактом LED+ на плате ЖКИ.

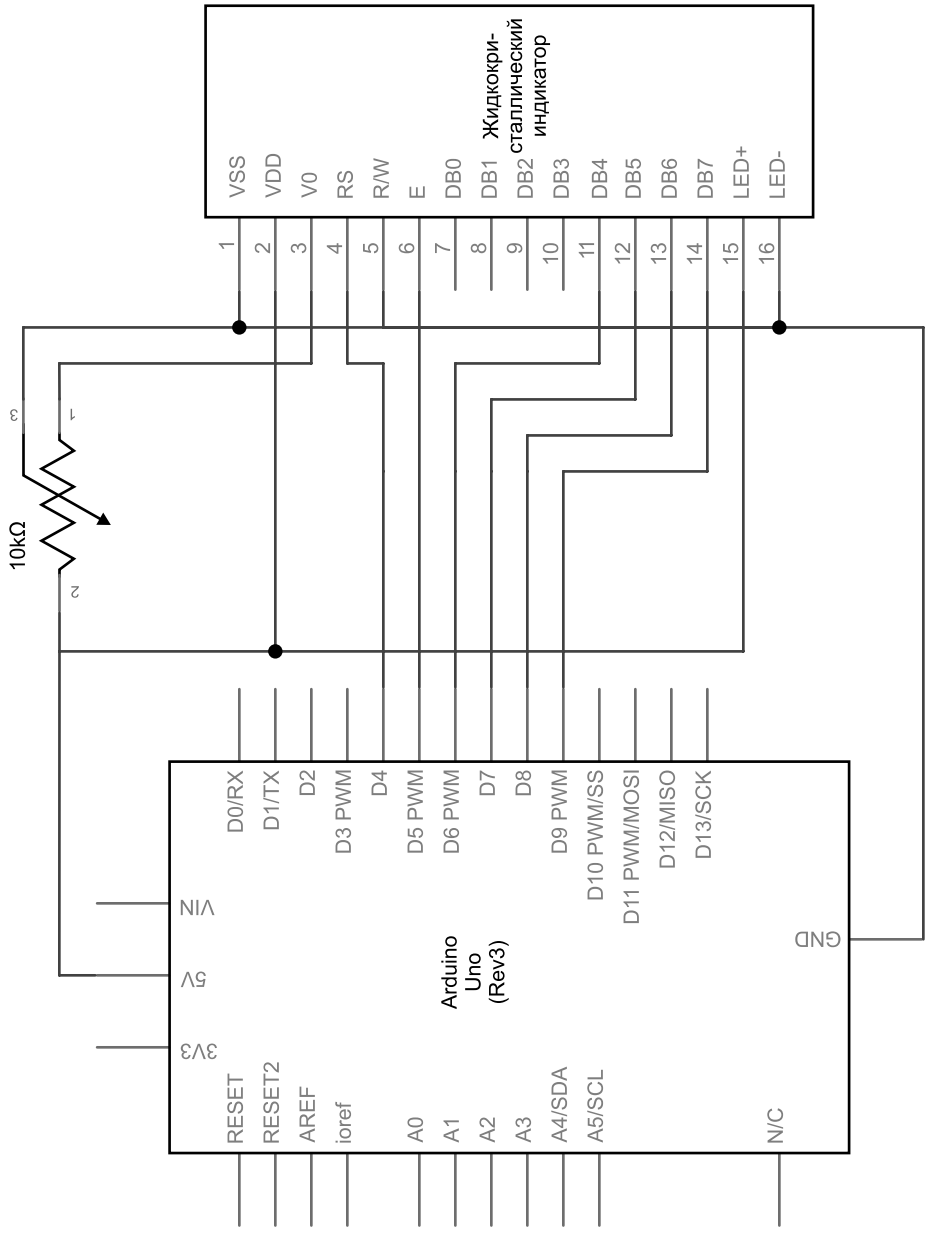


Рис. 9.2. Схема подключения жидкокристаллического индикатора

Использование символьного ЖКИ в скетче

Прежде чем задействовать символьный жидкокристаллический индикатор на рис. 9.1, познакомьтесь с некоторыми функциями, чтобы понять принцип их работы. Для этого рассмотрим несколько простых примеров. Но перед этим установите необходимую библиотеку Arduino из Library Manager (Менеджер библиотек) в IDE, как было описано в главе 7. Найдите и установите библиотеку LiquidCrystal by Arduino, Adafruit. Затем введите и загрузите скетч из листинга 9.1.

Листинг 9.1. Скетч, демонстрирующий работу с ЖКИ

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // К выводам RS, E, DB4, DB5, DB6, DB7

void setup()
{
  lcd.begin(16, 2);
  lcd.clear();
}

void loop()
{
  lcd.setCursor(0, 5);
  lcd.print("Hello");
  lcd.setCursor(1, 6);
  lcd.print("world!");
  delay(10000);
}
```

На рис. 9.3 показан результат работы скетча из листинга 9.1.

Рис. 9.3. Демонстрация ЖКИ: Hello world!

Теперь разложим по полочкам работу скетча из листинга 9.1. Прежде всего, скетч должен включать две начальные строки. Их цель — подключить библиотеку для работы с жидкокристаллическими индикаторами (она автоматически устанавливается из Arduino IDE) и сообщить ей, к каким контактам на плате Arduino подключен ЖКИ. Этой цели служат следующие две строки *перед* функцией `void setup()`:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // к выводам RS, E, DB4, DB5, DB6, DB7
```

Если понадобится использовать другие контакты на плате Arduino, измените их номера во второй строке.

Далее в функции `void setup()` библиотеке сообщается размер экрана ЖКИ в столбцах и строках. В этом скетче мы сообщаем, что экран ЖКИ имеет две строки по 16 символов в каждой:

```
lcd.begin(16, 2);
```

Отображение текста

После настройки ЖКИ в следующей строке производится очистка экрана:

```
lcd.clear();
```

Далее курсор устанавливается в начало вывода текста вызовом функции:

```
lcd.setCursor(x, y);
```

Здесь *x* — это номер столбца в строке (от 0 до 15), а *y* — номер строки (0 или 1). После этого текст выводится вызовом функции `lcd.print()`. К примеру, чтобы вывести слово `text`, скетч должен выполнить команду:

```
lcd.print("text");
```

Теперь, когда вы знаете, как размещать курсор и выводить текст, перейдем к отображению содержимого переменных.

Отображение переменных или чисел

Для выведения содержимого переменной на экран ЖКИ используйте вызов:

```
lcd.print(variable);
```

При выводе переменной типа `float` укажите количество десятичных знаков для вывода. `lcd.print(pi, 3)` в следующем примере отобразит значение `pi` с тремя десятичными знаками (рис. 9.4):

```
float pi = 3.141592654;
lcd.print("pi: ");
lcd.print(pi, 3);
```

Рис. 9.4. Отображение значения вещественной переменной на экран ЖКИ

Содержимое целочисленной переменой можно отобразить на экране ЖКИ не только в десятичном, но и в двоичном и шестнадцатеричном виде (листинг 9.2).

Листинг 9.2. Функции для отображения целых чисел в двоичном и шестнадцатеричном форматах

```
int zz = 170;
lcd.setCursor(0, 0);
lcd.print("Binary: ");
lcd.print(zz, BIN); // Вывести число 170
                    // в двоичном виде
lcd.setCursor(0, 1);
lcd.print("Hexadecimal: ");
lcd.print(zz, HEX); // Вывести число 170
                    // в шестнадцатеричном виде
```

Рис. 9.5. Результаты выполнения кода из листинга 9.2

Этот фрагмент выведет на экран ЖКИ текст (рис. 9.5).

Проект 28: определение собственных символов

Помимо букв, цифр и других стандартных символов, в каждом скетче можно определить до восьми собственных. Обратите внимание, что на экране ЖКИ каждый символ отображается в матрице, содержащей восемь рядов по пять точек *(пикселей)*. На рис. 9.6 эти матрицы показаны крупным планом.

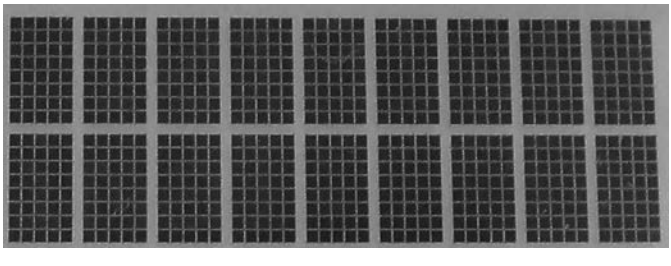


Рис. 9.6. Каждый символ состоит из восьми рядов по пять пикселей

Чтобы отобразить собственный символ, его сначала нужно определить в виде *массива*. Например, символ-смайлик определяется так:

```
byte a[8] = { B00000,
              B01010,
              B01010,
              B00000,
              B00100,
              B10001,
              B01110,
              B00000 };
```

Каждая цифра здесь соответствует пикселю: 0 — выключенному, 1 — включенному. Элементы массива представляют строки пикселей на экране. Первый элемент соответствует верхней строке, следующий — второй строке и т. д.

ПРИМЕЧАНИЕ

Если хотите использовать собственные символы, попробуйте сначала нарисовать их на разлинованной бумаге. Каждый закрашенный квадрат на ней будет соответствовать 1 в массиве, а каждый пустой — 0. По адресу https://maxpromer.github.io/LCD-Character-Creator/ вы найдете очень удобный редактор пользовательских символов, который сгенерирует готовый код для вас.

В этом примере первый элемент массива имеет значение `B00000`, поэтому все пиксели в верхнем ряду будут выключены. Во втором ряду (ему соответствует элемент `B01010`) будет включен каждый второй пиксель — они образуют верхние края глаз. Каждый следующий ряд продолжает заполнять символ.

Теперь нужно сохранить массив (определяющий новый символ) в первый из восьми слотов, предназначенных для нестандартных символов. Вот как это делается в функции `void setup()`:

```
lcd.createChar(0, a); // сохранить массив a[8] в слот 0
```

Для отображения символа нужно выполнить следующий вызов в `void loop()`:

```
lcd.write(byte(0));
```

Используем этот код для отображения нестандартных символов:

```
// Проект 28 — определение собственных символов
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // К выводам RS, E, DB4, DB5, DB6, DB7
byte a[8] = { B00000,
              B01010,
              B01010,
              B00000,
              B00100,
              B10001,
              B01110,
              B00000 };

void setup()
{
  lcd.begin(16, 2);
  lcd.createChar(0, a);
}
```

```
void loop()
{
  lcd.write(byte(0)); // вывести нестандартный символ из слота 0
                      // в следующую позицию курсора
}
```

Рисунок 9.7 демонстрирует два ряда смайликов на экране ЖКИ.

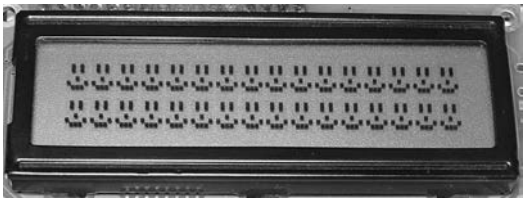


Рис. 9.7. Результат выполнения проекта 28

Символьные жидкокристаллические индикаторы универсальны и просты в использовании. С помощью такого индикатора можно создать цифровой термометр, объединив его с измерительной частью проекта 20 в главе 6. Но для отображения большого количества данных или графических элементов вам нужен *графический ЖКИ*.

Графические жидкокристаллические индикаторы

Графические жидкокристаллические индикаторы больше и дороже символьных, но они обладают большей гибкостью. Графические ЖКИ позволяют не только отображать текст, но и рисовать линии, точки, окружности и многое другое для создания визуальных эффектов. В проектах книги будет использоваться ST7735-совместимый графический ЖКИ с экраном 128 × 160 пикселей (рис. 9.8).

Рис. 9.8. Графический ЖКИ

Подключение графического ЖКИ

Прежде чем использовать графический ЖКИ, соедините его восемью проводами с платой Arduino. Это легко сделать, подключив провода со штекерами и гнездами к предварительно припаянной колодке с контактами на плате ЖКИ. Руководствуйтесь табл. 9.1.

Таблица 9.1. Карта соединений ЖКИ с платой Arduino

Контакт на плате ЖКИ	Контакт на плате Arduino	Назначение контакта на плате ЖКИ
Vcc	5 V	Напряжение питания
GND	GND	Питание («земля»)
CS	D10	Выбор
RST	D8	Сброс
A0 (или DC)	D9	Управление
SDA	D11	Входные данные
SCK	D13	Тактовые импульсы
LED	3.3 V	Питание подсветки

Использование ЖКИ

Прежде чем двинуться дальше, установите библиотеку поддержки графических ЖКИ из Library Manager (Менеджер библиотек) в IDE, как было описано в главе 7. Найдите и установите библиотеку TFT by Arduino, Adafruit.

Теперь, чтобы задействовать ЖКИ, вставьте следующие три строки перед функцией `void setup()`:

```
#include <TFT.h>       // Подключить поддержку графических ЖКИ
#include "SPI.h"       // Подключить библиотеку для работы с интерфейсом SPI

TFT TFTscreen = TFT(10, 9, 8); // Настроить выводы для работы с ЖКИ
```

(Пусть вас пока не смущают слова «с интерфейсом SPI» в комментариях. Сейчас эта строка — все, что нужно. Об интерфейсе SPI мы подробнее поговорим в главе 19.)

Теперь добавьте следующие строки в функцию `void setup()`, чтобы настроить индикатор:

```
TFTscreen.begin();             // Активировать ЖКИ
TFTscreen.background(0, 0, 0); // Очистить экран ЖКИ
```

Управление дисплеем

Вы можете выбрать один из пяти размеров отображения текста, как показано на рис. 9.9 и 9.10.

В первую очередь нужно настроить цвет фона для будущего изображения. Делается это так:

```
TFTscreen.background(b, g, r); // Установить цвет фона
```

Рис. 9.9. Четыре из пяти размеров текста, доступных в ЖКИ

Рис. 9.10. Наибольший, пятый размер текста из доступных в ЖКИ

Цвет фона задается в формате RGB — тремя числами, определяющими интенсивность трех базовых цветов (красного, зеленого, синего) в диапазоне от 0 до 255.

К примеру, белый фон можно получить, задав максимальную интенсивность всех трех цветов — 255, 255, 255. Чистый красный фон — задав интенсивность базового красного цвета равной 255, а синего и зеленого — 0. Для черного фона интенсивность всех трех цветов должна быть равна нулю (удобный список цветов и их представление в формате RGB вы найдете на https://www.rapidtables.com/web/color/RGB_Color.html).

Дальше нужно задать размер текста, если вы собираетесь выводить его в первый раз или если нужно изменить размер в середине скетча:

```
TFTscreen.setTextSize(x);
```

где *x* — число от 1 до 5, определяющее размер текста (рис. 9.9 и 9.10).

Далее вызовом следующей функции определяется цвет текста:

```
TFTscreen.stroke(B, G, R);
```

где *B*, *G* и *R* — значения интенсивности базовых цветов: синего, зеленого и красного.

Теперь можно вывести на экран сам текст:

```
TFTscreen.text("Hello, world!", x, y);
```

Этот вызов функции выведет текст «Hello, world!», начиная с позиции *x* и *y* на экране ЖКИ.

Такой способ отлично подходит для вывода статического текста. Но для вывода значения числовой переменной придется приложить немного больше усилий. Ее нужно преобразовать в символьный массив, размер которого соответствует максимально возможному значению. Если переменная предназначена для хранения значения, прочитанного с аналогового входа 0, и вам нужно вывести ее значение на экран, используйте такое объявление символьного массива:

```
char analogZero[4];
```

Далее в скетче перед выводом аналогового значения на экран ЖКИ преобразуйте значение в строку:

```
String sensorVal = String(analogRead(A0));
```

Скопируйте содержимое этой строки в символьный массив:

```
sensorVal.toCharArray(analogZero, 4);
```

И наконец, выведите символьный массив на экран командой `.text()`:

```
TFTscreen.text(analogZero, x, y);
```

которая выведет значение *analogZero*, начиная с позиции с координатами *x*, *y*.

Теперь используем все команды для вывода текста на экран ЖКИ в следующем проекте.

Проект 29: опробование текстовых функций в действии

В этом проекте мы отобразим текст на экране ЖКИ пяти разных размеров и значение, прочитанное с аналогового входа 0 платы Arduino.

Скетч

Соедините модуль ЖКИ с платой Arduino, руководствуясь табл. 9.1, и загрузите следующий скетч:

```
// Проект 29 – опробование текстовых функций в действии
#include <TFT.h> // Библиотека для работы с графическим ЖКИ
#include <SPI.h> // Библиотека для работы с интерфейсом SPI

TFT TFTscreen = TFT(10, 9, 8); // Назначить контакты цифровых сигналов
                               // для управления ЖКИ
char analogZero[4];

void setup()
{
  TFTscreen.begin();             // Активизировать ЖКИ,
  TFTscreen.background(0, 0, 0); // очистить экран, установив черный фон
}

void loop()
{
  TFTscreen.stroke(255, 255, 255);   // Белый текст
  TFTscreen.setTextSize(1);
  TFTscreen.text("Size One", 0, 0);
  TFTscreen.setTextSize(2);
  TFTscreen.text("Size Two", 0, 10);
  TFTscreen.setTextSize(3);
  TFTscreen.text("Size 3", 0, 30);
  TFTscreen.setTextSize(4);
  TFTscreen.text("Size 4", 0, 55);
  delay(2000);
  TFTscreen.background(0, 0, 0);     // Очистить экран, установив черный фон
  TFTscreen.setTextSize(5);
  TFTscreen.text("Five", 0, 0);
  delay(2000);
  TFTscreen.background(0, 0, 0);     // Очистить экран, установив черный фон
  TFTscreen.stroke(255, 255, 255);   // Белый текст
  TFTscreen.setTextSize(1);
```

```
  TFTscreen.text("Sensor Value :\n ", 0, 0);
  TFTscreen.setTextSize(3);
  String sensorVal = String(analogRead(A0));

  // Преобразовать измерение в символьный массив
  sensorVal.toCharArray(analogZero, 4);
  TFTscreen.text(analogZero, 0, 20);
  delay(2000);
  TFTscreen.background(0, 0, 0);     // Очистить экран, установив черный фон
}
```

Запуск скетча

Вы должны увидеть на экране текст всех пяти размеров — сначала первых четырех, а потом, после двухсекундной задержки, последнего пятого. Спустя еще две секунды должно появиться значение, прочитанное с аналогового входа 0 (рис. 9.11).

Рис. 9.11. Вывод значения с аналогового входа на экран ЖКИ

Создание более сложных изобразительных эффектов

Теперь рассмотрим несколько функций для рисования различных фигур. Помните, разрешение экрана графического ЖКИ — 160 пикселей по горизонтали и 128 по вертикали. Но при ссылке на координаты пикселей в наших скетчах их нужно указывать в диапазоне от 0 до 159 по горизонтали и от 0 до 127 по вертикали. Кроме того, как и в случае с примером вывода текста выше, нужно использовать пять строк кода, упомянутые в подразделе «Использование символьного ЖКИ в скетче» в начале главы, для инициализации дисплея.

Есть множество разных функций, позволяющих рисовать на экране ЖКИ точки (отдельные пиксели), линии, прямоугольники и окружности. Подумайте, чего вы хотите от проекта, и внесите немного творчества, чтобы создать красочное и полезное изображение на экране. Дальше мы познакомимся с этими функциями, и вы сможете увидеть их в действии с помощью демонстрационного скетча.

Перед тем как нарисовать любой объект, нужно определить его цвет. Это делается с помощью:

```
TFTscreen.stroke(B, G, R);
```

где *B*, *G* и *R* — значения интенсивности базовых цветов: синего, зеленого и красного соответственно.

Нарисовать одну точку на экране можно вызовом:

```
TFTscreen.point(X, Y);
```

где *X* и *Y* — горизонтальная и вертикальная координаты точки. Для данного ЖКИ координата *X* может принимать значения от 0 до 159, а координата *Y* — от 0 до 127.

Провести линию из одной точки в другую можно вызовом:

```
TFTscreen.line(X1, Y1, X2, Y2);
```

где *X1* и *Y1* — это координаты начальной точки, а *X2* и *Y2* — конечной.

Для рисования окружностей используется функция:

```
TFTscreen.circle(X, Y, R);
```

где *X* и *Y* — координаты центра окружности, а *R* — радиус в пикселях. Чтобы внутреннее пространство, ограниченное окружностью (или прямоугольником, рисование которого описывается ниже), залить цветом, перед вызовом функции `circle()` добавьте вызов:

```
TFTscreen.fill(B, G, R);
```

где *B*, *G* и *R* — значения интенсивности базовых цветов: синего, зеленого и красного, определяющих цвет заливки. Обратите внимание, что определение цвета заливки не изменяет цвета линий фигур. Поэтому перед рисованием фигур все еще нужно вызывать функцию `stroke()`.

Если вы собираетесь нарисовать несколько фигур одного цвета, достаточно вызвать функцию `fill()` только один раз. Чтобы выключить автоматическую заливку и вернуться к рисованию только границ фигур, нужно вызвать:

```
TFTscreen.noFill();
```

Нарисовать прямоугольник можно с помощью следующей функции:

```
TFTscreen.rect(X1, Y1, X2, Y2);
```

где *X1* и *Y1* — координаты левого верхнего угла прямоугольника, а *X2* и *Y2* — правого нижнего угла.

Проект 30: опробование графических функций в действии

Теперь задействуем все команды для вывода графики на экран ЖКИ.

Скетч

Соедините модуль ЖКИ с платой Arduino, руководствуясь табл. 9.1, и загрузите следующий скетч:

```
// Проект 30 — опробование графических функций в действии
#include <TFT.h> // Библиотека для работы с графическим ЖКИ
#include <SPI.h> // Библиотека для работы с интерфейсом SPI

TFT TFTscreen = TFT(10, 9, 8);   // Назначить контакты цифровых сигналов
                                 // для управления ЖКИ
int a;

void setup()
{
  TFTscreen.begin();             // Включить ЖКИ,
  TFTscreen.background(0, 0, 0); // очистить экран, установив черный фон,
  randomSeed(analogRead(0));     // инициализировать генератор случайных чисел
}

void loop()
{
  // Точки со случайными координатами
  for (a = 0; a < 100; a++)
  {
    TFTscreen.stroke(random(256), random(256), random(256));
    TFTscreen.point(random(160), random(120));
    delay(10);
  }
  delay(1000);
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный фон

  // Случайные линии
  for (a = 0; a < 100; a++)
  {
    TFTscreen.stroke(random(256), random(256), random(256));
    TFTscreen.line(random(160), random(120), random(160), random(120));
```

```
    delay(10);
  }
  delay(1000);
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный фон

  // Случайные окружности
  for (a = 0; a < 50; a++)
  {
    TFTscreen.stroke(random(256), random(256), random(256));
    TFTscreen.circle(random(160), random(120), random(50));
    delay(10);
  }
  delay(1000);
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный фон

  // Случайные окружности с заливкой
  for (a = 0; a < 50; a++)
  {
    TFTscreen.fill(random(256), random(256), random(256));
    TFTscreen.stroke(random(256), random(256), random(256));
    TFTscreen.circle(random(160), random(120), random(50));
    delay(10);
  }
  delay(1000);
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный фон

  // Случайные прямоугольники
  TFTscreen.noFill();
  for (a = 0; a < 50; a++)
  {
    TFTscreen.stroke(random(256), random(256), random(256));
    TFTscreen.rect(random(160), random(120), random(160), random(120));
    delay(10);
  }
  delay(1000);
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный фон

  // Случайные прямоугольники с заливкой
  TFTscreen.noFill();
  for (a = 0; a < 50; a++)
  {
    TFTscreen.fill(random(256), random(256), random(256));
    TFTscreen.stroke(random(256), random(256), random(256));
    TFTscreen.rect(random(160), random(120), random(160), random(120));
    delay(10);
  }
  delay(1000);
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный фон
}
```

После загрузки скетча на экране ЖКИ появятся результаты выполнения всех графических функций, рассмотренных нами в этой главе. Например, линии, как на рис. 9.12.

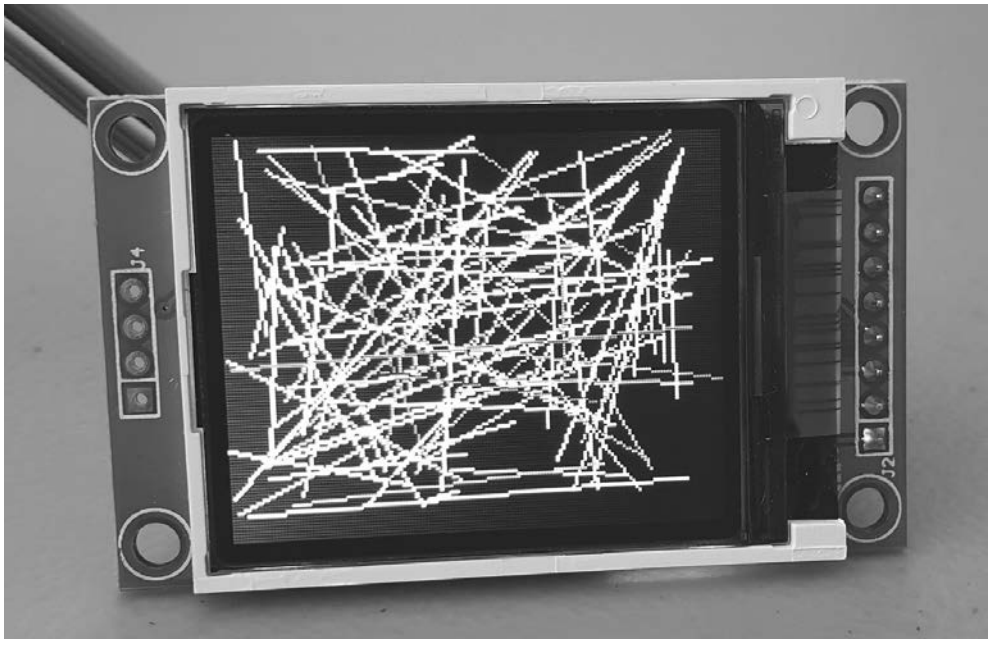


Рис. 9.12. Случайные линии на экране ЖКИ

С помощью этих функций и воображения можно создавать разные визуальные эффекты или отображать данные в виде графиков. В следующем разделе мы расширим наш проект термометра, использовав ЖКИ и некоторые из функций для работы с ним.

Проект 31: цифровой термометр с памятью

Наша цель в этом проекте — измерять температуру каждые 20 минут и отображать последние 120 замеров в виде графика. Каждый замер будет представлен пикселем с координатой y, соответствующей температуре, и с координатой x, соответствующей времени.

Самые свежие замеры будут слева, а график будет прокручиваться слева направо с каждым новым замером. Текущее значение температуры будет отображаться и в числовом виде.

Алгоритм

Этот проект только кажется сложным — на деле он довольно прост в реализации и требует всего двух функций. Первая принимает показания с датчика температуры TMP36 и сохраняет в массиве со 120 значениями. Каждый раз после выполнения нового замера результаты предыдущих 119 смещаются в массиве вниз, чтобы освободить место для новых данных. Самые старые значения затираются следующими за ними.

Вторая функция создает изображение на экране ЖКИ. Ее задача — отобразить текущую температуру, определить масштаб для графика и вывести каждый замер в виде пикселя на экране для получения графика изменения температуры во времени.

Оборудование

Ниже перечислено оборудование для этого проекта:

- один графический ЖК-индикатор с размером экрана 160 × 128 пикселей, совместимый с интерфейсом ST7735, описанный выше в этой главе;
- один датчик температуры TMP36;
- несколько отрезков провода разной длины;
- одна макетная плата;
- плата Arduino и кабель USB.

Подключите графический ЖКИ в соответствии с табл. 9.1 и соедините датчик TMP36 с контактами `5 V`, `A5` и `GND` на плате Arduino, как описано в проекте 20 главы 6.

Скетч

Наш скетч сочетает код, который мы использовали для измерения температуры в главе 6, и графические функции из этой главы. Введите и загрузите следующий скетч с комментариями, описывающими его работу.

```
// Проект 31 — цифровой термометр с памятью

#include <TFT.h> // Библиотека для работы с графическим ЖКИ
#include <SPI.h> // Библиотека для работы с интерфейсом SPI

TFT TFTscreen = TFT(10, 9, 8); // Назначить выводы цифровых сигналов
                               // для управления ЖКИ
int tcurrent = 0;
int tempArray[120];

char currentString[3];
```

```
void getTemp() // Функция чтения температуры с датчика TMP36
{
  float sum = 0;
  float voltage = 0;
  float sensor = 0;
  float celsius;

  // Прочитать значение с датчика и преобразовать
  // его в градусы Цельсия
  sensor = analogRead(5);
  voltage = (sensor * 5000) / 1024;
  voltage = voltage - 500;
  celsius = voltage / 10;
  tcurrent = int(celsius);

  // Вставить новый замер в начало массива
  for (int a = 119 ; a >= 0 ; --a )
  {
    tempArray[a] = tempArray[a - 1];
  }
  tempArray[0] = tcurrent;
}

void drawScreen() // Выводит изображение на экран ЖКИ
{
  int q;

  // Вывести текущую температуру
  TFTscreen.background(0, 0, 0);   // Очистить экран, установив черный цвет фона
  TFTscreen.stroke(255, 255, 255); // Белый текст
  TFTscreen.setTextSize(2);
  TFTscreen.text("Current:", 20, 0);
  String tempString = String(tcurrent);
  tempString.toCharArray(currentString, 3);
  TFTscreen.text(currentString, 115, 0);

  // Нарисовать оси графика
  TFTscreen.setTextSize(1);
  TFTscreen.text("50", 0, 20);
  TFTscreen.text("45", 0, 30);
  TFTscreen.text("40", 0, 40);
  TFTscreen.text("35", 0, 50);
  TFTscreen.text("30", 0, 60);
  TFTscreen.text("25", 0, 70);
  TFTscreen.text("20", 0, 80);
  TFTscreen.text("15", 0, 90);
  TFTscreen.text("10", 0, 100);
  TFTscreen.text(" 5", 0, 110);
  TFTscreen.text(" 0", 0, 120);
  TFTscreen.line(20, 20, 20, 127);
```

```
  // Нарисовать график изменения температуры
  for (int a = 25 ; a < 145 ; a++)
  {
    // Преобразовать температуру в координату Y на экране ЖКИ
    q = (123 - (tempArray[a - 25] * 2));
    TFTscreen.point(a, q);
  }
}

void setup()
{
  TFTscreen.begin();              // Включить ЖКИ
  TFTscreen.background(0, 0, 0); // Очистить экран, установив черный цвет фона
}

void loop()
{
  getTemp();
  drawScreen();
  for (int a = 0 ; a < 20 ; a++) // Ждать 20 минут до следующего замера
  {
    delay(60000); // Ждать 1 минуту
  }
}
```

Результат

В результате на экране должно сформироваться изображение, напоминающее показанное на рис. 9.13.

Рис. 9.13. Результаты работы проекта 31

Доработка скетча

Некоторые из нас лучше воспринимают информацию в графическом, а не в числовом виде. Поэтому иногда лучше отобразить данные с помощью линейных графиков или гистограмм.

Такой проект можно использовать для отображения самых разных данных, например уровней напряжения от разных датчиков, измеряемых на контактах аналоговых входов. Можно добавить еще один датчик температуры и отображать сразу два графика. Почти все, что имеет числовое выражение, можно отобразить на экране графического ЖКИ.

Что дальше?

Теперь, после знакомства с жидкокристаллическими индикаторами, стало понятнее, что плата Arduino — это маленький компьютер: она может принимать и обрабатывать данные и отображать результаты. Но это только начало. В следующей главе вы узнаете, как создаются библиотеки, напишете свою собственную и будете использовать ее вместе с датчиком температуры из предыдущих проектов.

10

Создание своих библиотек для Arduino

В этой главе вы:

- познакомитесь с устройством библиотек для Arduino;
- создадите простую библиотеку для повторного применения;
- узнаете, как установить свою библиотеку в Arduino IDE;
- создадите библиотеку, принимающую значения для выполнения функции;
- создадите библиотеку, обрабатывающую данные с датчика и возвращающую готовые для использования значения.

Вспомните проект 22 из главы 7, где мы установили библиотеку с функциями для сохранения данных на SD-карту. Использование библиотеки помогло сократить время на разработку скетча — нам не пришлось писать функции для выполнения операций с модулем карты.

В дальнейшем, создавая скетчи для решения ваших задач, вы будете неоднократно использовать одни и те же функции, написанные вами. Для таких случаев разумно создать свою библиотеку, которую можно было бы установить и использовать в скетчах.

В этой главе вы узнаете, как организовать функции в библиотеку. Примеры отсюда покажут вам все, что нужно знать для создания собственных библиотек. Итак, приступим.

Создание первой библиотеки для Arduino

Для начала рассмотрим листинг 10.1. Он содержит две функции, `blinkSlow()` и `blinkFast()`, которые используются для медленного и быстрого мигания встроенным светодиодом Arduino.

Листинг 10.1. Мигание встроенным светодиодом Arduino

```
void setup()
{
  pinMode(13, OUTPUT); // Использовать встроенный светодиод
}

void blinkSlow()
{
  for (int i = 0; i < 5; i++)
  {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  }
}

void blinkFast()
{
  for (int i = 0; i < 5; i++)
  {
    digitalWrite(13, HIGH);
    delay(250);
    digitalWrite(13, LOW);
    delay(250);
  }
}

void loop()
{
  blinkSlow();
  delay(1000);
  blinkFast();
  delay(1000);
}
```

Если в скетче вам понадобятся функции `blinkSlow()` и `blinkFast()`, а библиотеки у вас нет, вам придется снова вводить их вручную. Но если поместить код функций в библиотеку, то вы сможете вызывать их, просто подключив ее в начале скетча.

Устройство библиотеки для Arduino

Библиотека для Arduino состоит из трех файлов и нескольких необязательных скетчей с примерами, показывающими использование библиотеки. Вот три необходимых файла для каждой библиотеки Arduino:

`<`*`название_библиотеки`*`>.h` — заголовочный файл;

`<`*`название_библиотеки`*`>.cpp` — файл с исходным кодом;

`KEYWORDS.TXT` — определения ключевых слов.

В первых двух именах файлов замените `<`*`название_библиотеки`*`>` фактическим именем своей библиотеки. Давайте назовем нашу первую библиотеку *blinko*. Так первые два файла получат имена `blinko.h` и `blinko.cpp`.

Заголовочный файл

Файл `blinko.h` — это *заголовочный файл*. Так он называется потому, что содержит определения (заголовки) функций, переменных и других компонентов внутри библиотеки. Заголовочный файл для библиотеки blinko показан в листинге 10.2.

Листинг 10.2. Заголовочный файл библиотеки blinko

```
  /*
❶ blinko.h – библиотека с функциями для мигания встроенным светодиодом
             на плате Arduino, подключенным к выводу D13
  */
❷ #ifndef blinko_h
  #define blinko_h
❸ #include "Arduino.h" // Открывает библиотеке доступ к стандартным типам
                       // и константам Arduino
❹ class blinko         // Функции и переменные, определяемые библиотекой
  {
    public:
      blinko();
      void slow();
      void fast();
  };
❺ #endif
```

Этот файл немного похож на типичный скетч для Arduino, но есть и некоторые отличия. В ❶ находится комментарий, описывающий назначение библиотеки. Это не обязательно, но их лучше добавлять, чтобы облегчить использование библиотеки для других.

В строке ❷ код проверяет, была ли библиотека уже подключена в главном файле скетча. В строке ❸ подключается стандартная библиотека Arduino, позволяющая библиотеке blinko использовать стандартные функции, типы и константы Arduino.

В строке ❹ объявляется класс. Его можно представлять как коллекцию, объединяющую все переменные и функции библиотеки, включая имя самой библиотеки. Внутри класса могут быть общедоступные переменные и функции для использования в скетче, подключившем библиотеку, а также частные переменные и функции, доступные только внутри класса.

У каждого класса есть *конструктор*, который используется для создания экземпляра класса. Его имя совпадает с именем класса. Такая организация может показаться сложной. Но, просмотрев примеры в этой главе и создав несколько своих библиотек, вы поймете эти конструкции.

Внутри нашего класса есть конструктор `blinko()` и две библиотечные функции: `slow()` и `fast()`. Они следуют за ключевым словом `public:`. Это значит, что их сможет использовать любой скетч, подключивший библиотеку blinko.

Наконец, в строке ❺ завершается определение заголовка. Обертывая определение заголовка директивами `#if`/`#endif`, мы гарантируем, что он не будет загружен дважды.

Файл с исходным кодом

Теперь заглянем в `blinko.cpp`. Файлы с расширением `.cpp` содержат исходный код, который будет выполняться при использовании библиотеки. Файл с исходным кодом библиотеки blinko приводится в листинге 10.3.

Листинг 10.3. Файл с исходным кодом библиотеки blinko

```
  /*
❶ blinko.cpp – библиотека с функциями для мигания встроенным светодиодом
               на плате Arduino, подключенным к выводу D13
  */
  ❷ #include "Arduino.h" // Открывает библиотеке доступ к стандартным типам
                         // и константам Arduino
  #include "blinko.h"

❸ blinko::blinko() // Выполняет операции в момент активации библиотеки
  {
    pinMode(13, OUTPUT);
  }

❹ void blinko::slow()
  {
    for (int i=0; i<5; i++)
    {
      digitalWrite(13, HIGH);
```

```
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
  }
}

❹ void blinko::fast()
{
  for (int i=0; i<5; i++)
  {
    digitalWrite(13, HIGH);
    delay(250);
    digitalWrite(13, LOW);
    delay(250);
  }
}
```

Файл с исходным кодом содержит реализации библиотечных функций, которые мы предполагаем использовать в разных скетчах. Здесь применяются и новые конструктивные элементы. В ❷ мы открываем для нашей библиотеки доступ к стандартным функциям, типам и константам Arduino и определениям в нашем заголовочном файле.

В ❸ находится определение функции-конструктора. Конструктор реализует операции, которые должны выполняться перед использованием библиотеки. В этом примере он настраивает цифровой контакт 13 на работу в качестве вывода, чтобы дать возможность управлять встроенным светодиодом Arduino.

Начиная с ❹, следуют определения функций, включенных в библиотеку. Они похожи на функции в обычном скетче с одним важным отличием: их определения начинаются с имени класса библиотеки и двух двоеточий. Например, вместо `void fast()` мы определили имя функции как `void blinko::fast()`.

Файл KEYWORDS.TXT

Теперь нужно создать файл `KEYWORDS.TXT`. Среда разработки Arduino IDE использует этот файл для определения ключевых слов в библиотеке и выделяет их в редакторе IDE. В листинге 10.4 показано содержимое файла `KEYWORDS.TXT` для нашей библиотеки blinko.

Листинг 10.4. Файл с ключевыми словами библиотеки blinko

```
blinko          KEYWORD1
slow            KEYWORD2
fast            KEYWORD2
```

Первая строка — это имя библиотеки, оно отмечено как `KEYWORD1`. Имена обеих функций обозначены как `KEYWORD2`. Обратите внимание, что пространство между

ключевыми словами и их обозначениями должно оформляться нажатием клавиши Tab, а не Пробел.

Сейчас у нас есть все три файла для создания библиотеки. Было бы неплохо включить и скетч с примерами, чтобы пользователи могли понять, как правильно применять функции. В листинге 10.5 показан наш скетч с примерами использования библиотеки blinko.

Листинг 10.5. Скетч с примерами использования библиотеки blinko

```
❶ #include <blinko.h>

❷ blinko ArduinoLED;
  void setup() {
  }

  void loop()
  {
❸   ArduinoLED.slow(); // Медленное мигание светодиодом,
                      // одна вспышка в секунду
    delay(1000);
❹   ArduinoLED.fast(); // Быстрое мигание светодиодом,
                      // четыре вспышки в секунду
    delay(1000);
}
```

Скетч очень простой. Он показывает, как использовать функции `slow()` и `fast()` в нашей библиотеке. Все, что конечному пользователю нужно сделать после установки библиотеки, — подключить ее ❶, создать экземпляр класса ❷ и вызвать любую функцию, как показано в ❸ и ❹.

Установка новой библиотеки

Теперь, чтобы поделиться новой библиотекой для Arduino с другими пользователями, нужно создать ZIP-файл. В дальнейшем с помощью этого файла они смогут установить библиотеку, как рассказывалось в главе 7.

Создание ZIP-файла в Windows версии 7 и выше

Чтобы создать ZIP-файл в Windows, следуйте инструкциям ниже.

Сначала поместите три файла библиотеки и скетч с примерами (хранящийся в отдельной папке, как и все скетчи) в одно место, как показано на рис. 10.1.

Выберите все файлы, щелкните правой кнопкой мыши на любом из них и найдите в контекстном меню пункт Send To ▸ Compressed (Zipped) Folder (Отправить ▸ Сжатая ZIP-папка), как показано на рис. 10.2.

Рис. 10.1. Файлы нашей библиотеки в одной папке

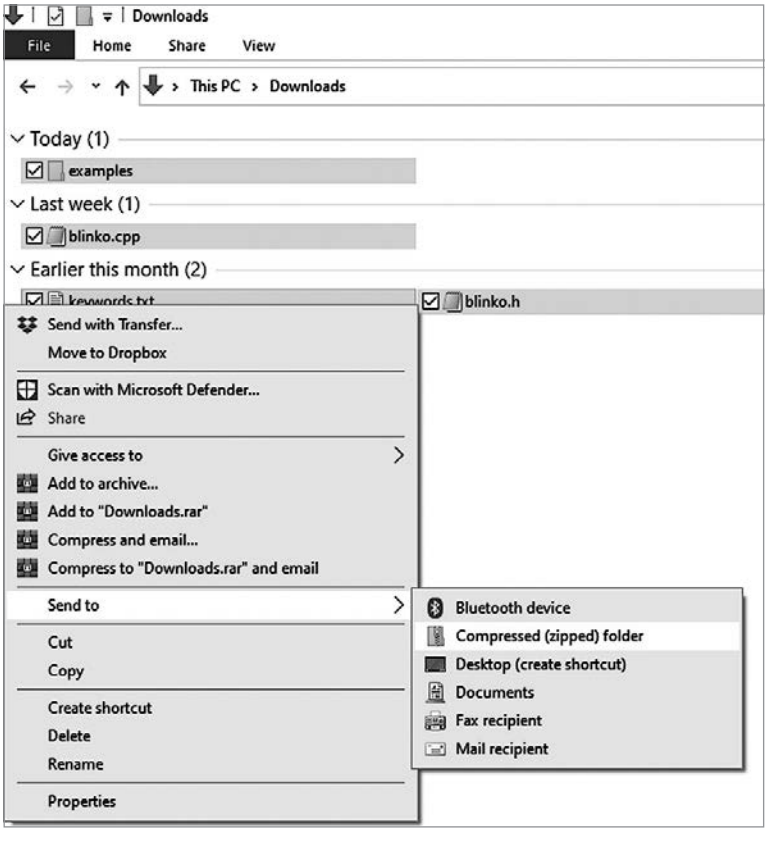


Рис. 10.2. Упаковка файлов библиотеки в архив

В папке появится новый файл с расширением `.zip` и с именем, доступным для редактирования. Измените имя на *blinko* и нажмите Enter (рис. 10.3).

Теперь можете перейти к разделу «Установка новой библиотеки» ниже в этой главе.

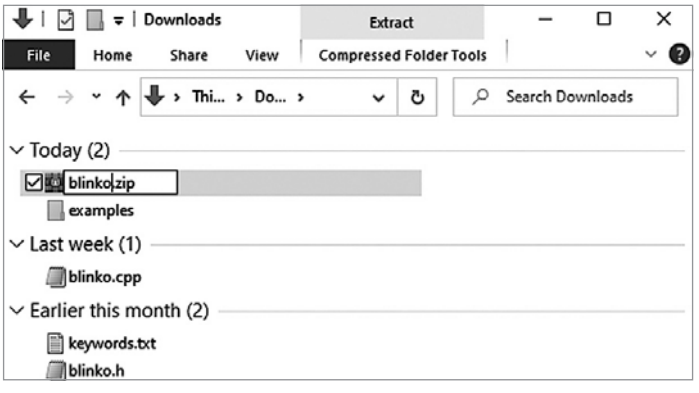


Рис. 10.3. Изменение имени ZIP-файла библиотеки

Создание ZIP-файла в Mac OS версии X и выше

Чтобы создать ZIP-файл в Mac OS X, поместите три файла библиотеки и скетч с примерами (хранящийся в отдельной папке, как и все скетчи) в одно место, как показано на рис. 10.4.

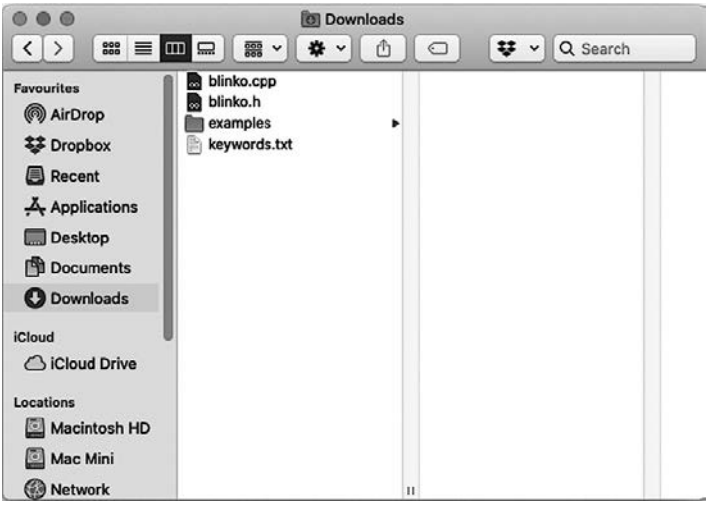


Рис. 10.4. Файлы библиотеки в одной папке

Выберите все файлы, щелкните правой кнопкой мыши на любом из них и найдите в контекстном меню пункт Compress 4 Items (Сжать 4 элемента) (рис. 10.5).

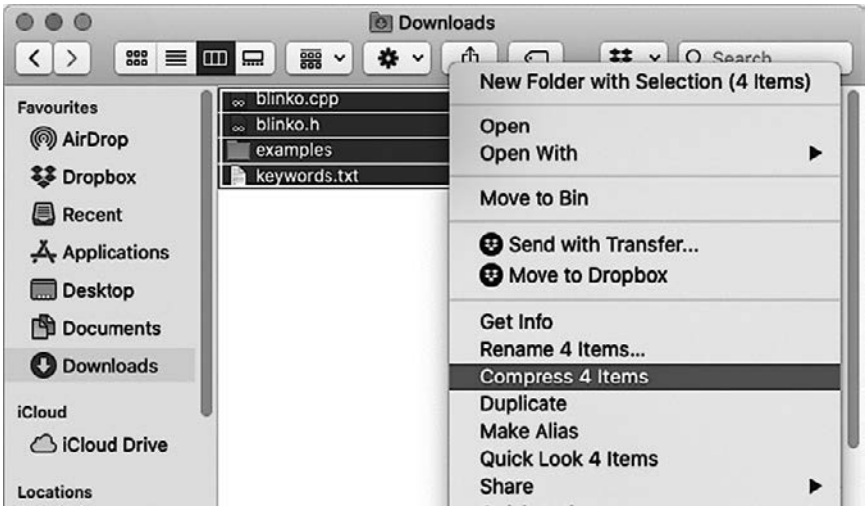


Рис. 10.5. Упаковка файлов библиотеки в архив

Через несколько секунд в папке появится новый файл с именем `Archive.zip` (рис. 10.6).

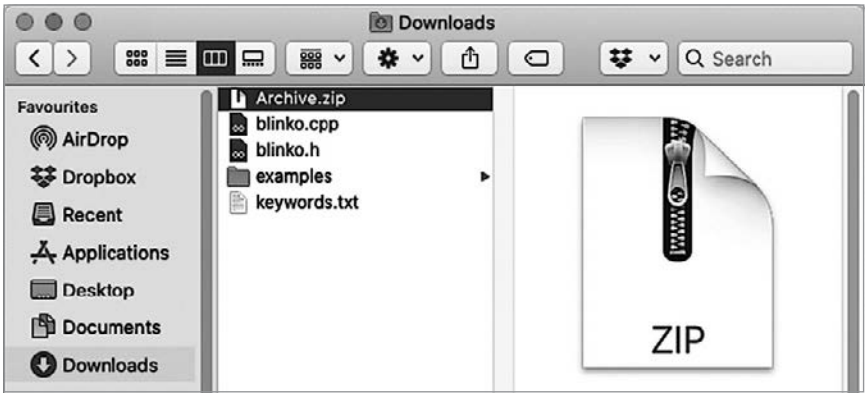


Рис. 10.6. Архив с файлами библиотеки

Щелкните на файле `Archive.zip` и измените его имя на `blinko.zip` (рис. 10.7).

Теперь у вас есть ZIP-файл с библиотекой. Вы можете передать его другим или установить у себя.

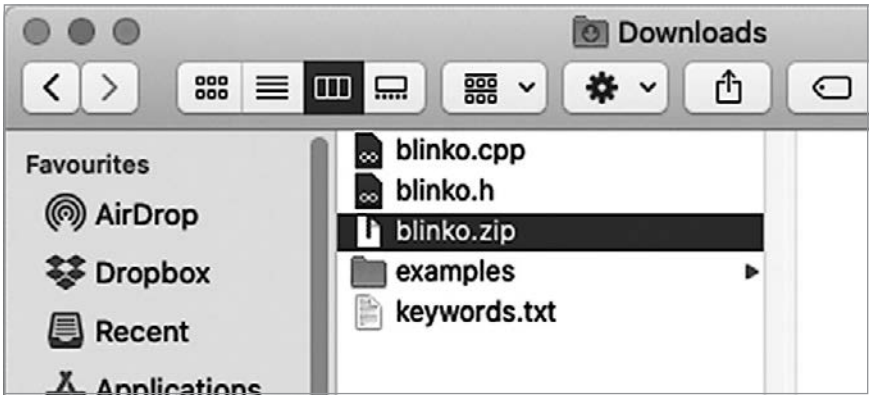


Рис. 10.7. Архив с файлами библиотеки после переименования

Установка новой библиотеки

После получения ZIP-файла можно установить библиотеку, как описывалось в разделе «Загрузка библиотеки в виде ZIP-файла» в главе 7. Установив библиотеку и перезапустив Arduino IDE, выберите в меню пункт Sketch ▸ Include Library (Скетч ▸ Подключить библиотеку), и вы увидите свою библиотеку в списке (рис. 10.8).

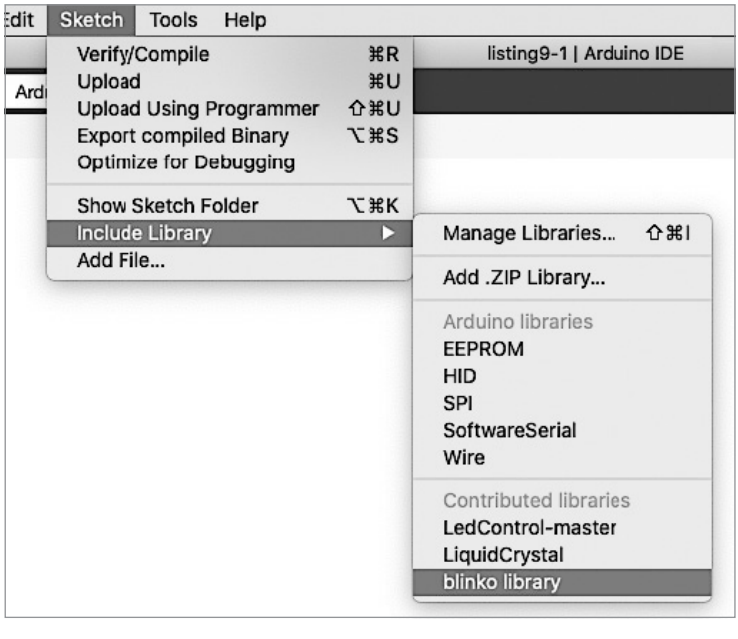


Рис. 10.8. Свежеустановленная библиотека доступна в Arduino IDE

Вы с легкостью можете открыть скетч с примерами. Выберите в меню пункт File ▸ Examples ▸ blinko (Файл ▸ Примеры ▸ blinko), как показано на рис. 10.9.

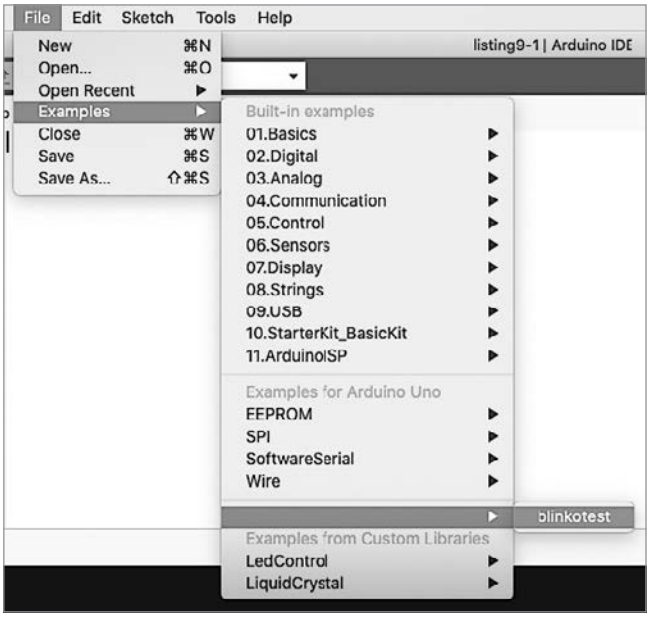


Рис. 10.9. Скетч с примерами использования библиотеки тоже был установлен

Создание библиотеки, принимающей значения для выполнения функции

Теперь мы знаем, как создать простую библиотеку для Arduino. Поэтому продвинемся дальше и создадим библиотеку, принимающую некоторое значение и действующую в соответствии с ним. И снова мы начнем с примера функции, а после преобразуем ее в библиотеку.

Взгляните на скетч в листинге 10.6. Он использует функцию `void blinkType()`, принимающую два параметра: количество миганий встроенным светодиодом и продолжительность периода включения/выключения.

Листинг 10.6. Скетч, демонстрирующий функцию blinkType()

```
void setup() {
  pinMode(13, OUTPUT); // Использовать встроенный светодиод
}
```

```
void blinkType(int blinks, int duration)
// blinks — число миганий
// duration — продолжительность вспышки в миллисекундах
{
  for (int i = 0; i < blinks; i++)
  {
    digitalWrite(13, HIGH);
    delay(duration);
    digitalWrite(13, LOW);
    delay(duration);
  }
}

void loop()
{
  // Мигнуть светодиодом десять раз с продолжительностью каждой вспышки 250 мс
  blinkType(10, 250);
  delay(1000);
  // Мигнуть светодиодом три раза с продолжительностью каждой вспышки 1 с
  blinkType(3, 1000);
  delay(1000);
}
```

Здесь функция `void blinkType()` принимает два значения и действует в соответствии с ними. Первое — это количество включений/выключений встроенного светодиода, а второе — время задержки в миллисекундах для каждой вспышки.

Превратим эту функцию в библиотеку с именем blinko2. В листинге 10.7 показан заголовочный файл для нее.

Листинг 10.7. Заголовочный файл библиотеки blinko2

```
/*
  blinko2.h — мигает встроенным светодиодом, подключенным к выводу D13,
              принимает число миганий и продолжительность вспышки
              в миллисекундах
*/

#ifndef blinko2_h
#define blinko2_h

#include "Arduino.h"

class blinko2
{
  public:
    blinko2();
    void blinkType(int blinks, int duration);
```

```
❶ private:
    int blinks;
    int duration;
};
#endif
```

Этот файл имеет ту же структуру, что и заголовочный файл оригинальной библиотеки blinko. Но в нем появился новый частный раздел `private` ❶. Переменные, объявленные в таком разделе, предназначены для внутреннего использования библиотекой и недоступны из скетча. В листинге 10.8 показано, как они используются в файле с исходным кодом библиотеки.

Листинг 10.8. Файл с исходным кодом библиотеки blinko2

```
/*
  blinko2.cpp – мигает встроенным светодиодом, подключенным к выводу D13,
                принимает число миганий и продолжительность вспышки
                в миллисекундах
*/

#include "Arduino.h"
#include "blinko2.h"

blinko2::blinko2()
{
❶   pinMode(13, OUTPUT);
}

❷ void blinko2::blinkType(❸ int blinks, ❹ int duration)
{
  for (int i=0; i<blinks; i++)
  {
    digitalWrite(13, HIGH);
    delay(duration);
    digitalWrite(13, LOW);
    delay(duration);
  }
}
```

Файл с исходным кодом blinko2 имеет ту же структуру, что и файл оригинальной библиотеки blinko.

Мы настраиваем контакт 13 на работу в качестве цифрового вывода ❶. В ❷ определяем функцию `blinkType()`, принимающую количество миганий ❸ и время задержки ❹. Действие библиотеки можно наблюдать в скетче с примером в листинге 10.9.

Листинг 10.9. Скетч с примером использования библиотеки blinko2

```
#include <blinko2.h>

blinko2 ArduinoLED;

void setup() {}

void loop()
{
  // Мигнуть светодиодом три раза с продолжительностью каждой вспышки 250 мс
  ArduinoLED.blinkType(3,250);
  delay(1000);

  // Мигнуть светодиодом десять раз с продолжительностью каждой вспышки 1 с
  ArduinoLED.blinkType(10,1000);
  delay(1000);
}
```

Теперь нужно создать файл с ключевыми словами для новой библиотеки blinko2. Не забудьте, что они отделяются от обозначений табуляцией, а не пробелами. Вот содержимое файла `KEYWORDS.TXT`:

```
blinko2       KEYWORD1
blinkType     KEYWORD2
```

Теперь создайте ZIP-файл и установите библиотеку, как описывалось выше. Затем откройте и запустите скетч с примерами blinko2, чтобы увидеть, как она работает.

Создание библиотеки, обрабатывающей и отображающей прочитанные с датчиков значения

В последнем примере создания библиотек для Arduino вернемся к датчику температуры TMP36 из предыдущих проектов. Наша библиотека ArduinoTMP36 будет читать значение из датчика TMP36 и выводить температуру в градусах Цельсия и Фаренгейта на монитор порта.

Сначала подключите TMP36 к Arduino, следуя схеме на рис. 10.10.

В листинге 10.10 приводится скетч, который мы превратим в библиотеку. Он определяет две функции: `readC()` и `readF()`. Они читают показания датчика TMP36 через аналоговый вывод `0`, преобразуют их в градусы Цельсия и Фаренгейта и возвращают результат.

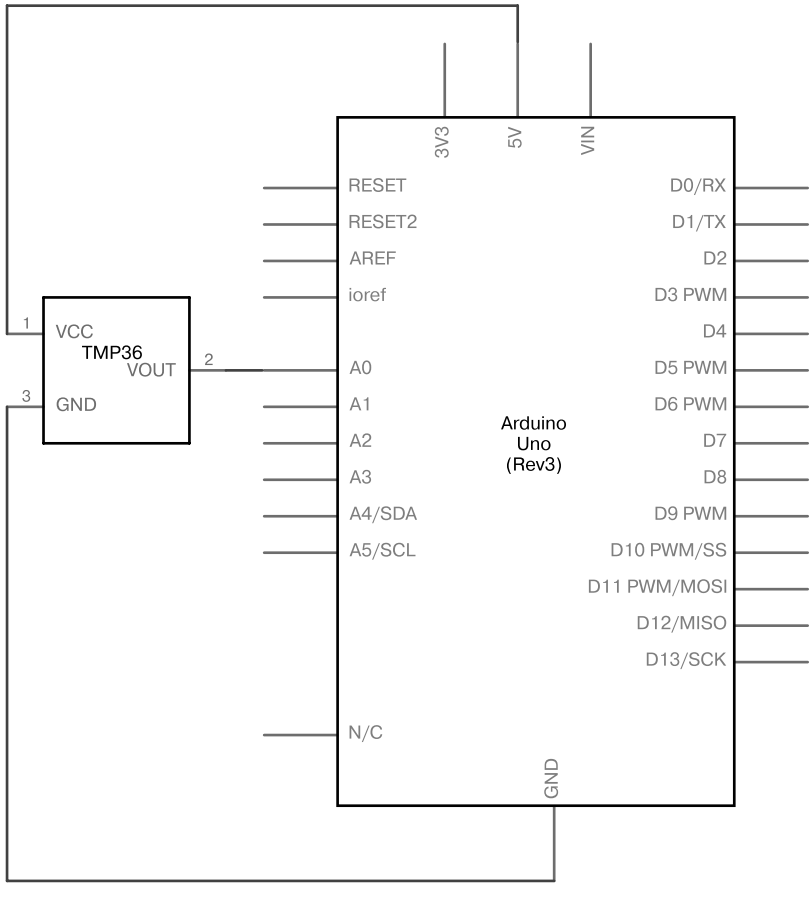


Рис. 10.10. Схема для демонстрации библиотеки ArduinoTMP36

Листинг 10.10. Скетч, демонстрирующий работу с TMP36

```
// Выводит температуру, прочитанную с датчика TMP36,
// в градусах Цельсия и Фаренгейта

float temperature;

float readC()
{
  float tempC;
  tempC = analogRead(0);
  tempC = tempC = (tempC * 5000) / 1024;
  tempC = tempC - 500;
  tempC = tempC / 10;
```

```
  return tempC;
}

float readF()
{
  float tempC;
  float tempF;
  tempC = analogRead(0);
  tempC = tempC = (tempC * 5000) / 1024;
  tempC = tempC - 500;
  tempC = tempC / 10;
  tempF = (tempC * 1.8) + 32;
  return tempF;
}

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

void loop()
{
  Serial.print("Temperature in Celsius is: ");
  temperature = readC();
  Serial.println(temperature);
  Serial.print("Temperature in Fahrenheit is: ");
  temperature = readF();
  Serial.println(temperature);
  delay(1000);
}
```

Функции для преобразования температуры — идеальные кандидаты для включения в библиотеку, которую мы назовем ArduinoTMP36. Заголовочный файл библиотеки показан в листинге 10.11.

Листинг 10.11. Заголовочный файл библиотеки ArduinoTMP36

```
❶ #ifndef ArduinoTMP36_h
  #define ArduinoTMP36_h

  #include "Arduino.h"

  class ArduinoTMP36
  {
❷   public:
      ArduinoTMP36();
      float readC();
      float readF();
❸   private:
```

```
    float tempC;
    float tempF;
};
#endif
```

Вы наверняка узнали уже знакомую структуру заголовочного файла. В ❶ мы исключаем возможность повторной загрузки библиотеки. Внутри класса ❷ мы объявляем общедоступные элементы, в число которых входят конструктор и функции `readC()` и `readF()`. В ❸ мы объявляем частные компоненты: две переменные, используемые библиотекой.

В листинге 10.12 приводится файл с исходным кодом библиотеки.

Листинг 10.12. Файл с исходным кодом библиотеки ArduinoTMP36

```
#include "Arduino.h"
#include "ArduinoTMP36.h"

ArduinoTMP36::ArduinoTMP36()
{
}

float ArduinoTMP36::readC()
{
  float tempC;
  tempC = analogRead(0);
  tempC = tempC=(tempC*5000)/1024;
  tempC = tempC-500;
  tempC = tempC/10;
  return tempC;
}

float ArduinoTMP36::readF()
{
  float tempC;
  float tempF;
  tempC = analogRead(0);
  tempC = tempC=(tempC*5000)/1024;
  tempC = tempC-500;
  tempC = tempC/10;
  tempF = (tempC*1.8)+32;
  return tempF;
}
```

Файл с исходным кодом содержит две функции, вычисляющие значения температур. Они определены как возвращающие значение типа `float`, потому что

возвращают вещественные числа. Вычисления производятся по тем же формулам, что и в проекте 8 главы 4.

Наконец, для нашей новой библиотеки ArduinoTMP36 нужно создать файл ключевых слов (не забывайте использовать табуляцию, а не пробелы). Ниже показано содержимое файла `KEYWORDS.TXT`:

```
ArduinoTMP36   KEYWORD1
readC          KEYWORD2
readF          KEYWORD2
```

Теперь создайте ZIP-файл и установите библиотеку, как описывалось выше. Затем откройте и запустите скетч из листинга 10.13 с примерами использования библиотеки ArduinoTMP36.

Листинг 10.13. Скетч с примерами использования библиотеки ArduinoTMP36

```
❶ #include <ArduinoTMP36.h>

  ArduinoTMP36 thermometer;

❷ float temperature;

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

  void loop()
  {
    Serial.print("Temperature in Celsius is: ");
❸   temperature=thermometer.readC();
    Serial.println(temperature);
    Serial.print("Temperature in Fahrenheit is: ");
❹   temperature=thermometer.readF();
    Serial.println(temperature);
    delay(1000);
}
```

Скетч просто подключает библиотеку в ❶ и создает экземпляр. Потом он объявляет переменную ❷ для сохранения полученной от библиотеки температуры. Далее скетч запрашивает температуру в градусах Цельсия ❸ и Фаренгейта ❹.

Откройте окно Serial Monitor (Монитор порта) и установите скорость передачи данных 9600 бод. После этого в окне должен появиться обновляемый прокручиваемый список значений текущей температуры в градусах Цельсия и Фаренгейта (рис. 10.11).

```
Temperature in Fahrenheit is: 63.29
Temperature in Celsius is: 16.89
Temperature in Fahrenheit is: 63.29
Temperature in Celsius is: 16.89
Temperature in Fahrenheit is: 63.29
Temperature in Celsius is: 16.89
Temperature in Fahrenheit is: 63.29
Temperature in Celsius is: 16.89
Temperature in Fahrenheit is: 63.29
Temperature in Celsius is: 16.89
Temperature in Fahrenheit is: 62.41
Temperature in Celsius is: 16.89
Temperature in Fahrenheit is: 62.41
```

Рис. 10.11. Пример вывода значений, полученных с помощью библиотеки ArduinoTMP36

Теперь вы можете оценить, сколько времени и строк кода экономится за счет использования библиотеки вместо добавления функций вручную.

Что дальше?

Теперь вы сможете сами создавать библиотеки для Arduino. Это поможет вам лучше понять библиотеки из других источников. Можете попрактиковаться и создать библиотеки для проектов из этой книги, которые вы уже опробовали.

В следующей главе вы узнаете, как читать ввод пользователя с цифровой клавиатуры. Переверните страницу — и приступим.

11

Цифровые клавиатуры

В этой главе вы:

- узнаете, как подключить цифровую клавиатуру к плате Arduino;
- научитесь читать в скетчах значения, набираемые на клавиатуре;
- освоите прием принятия решений с применением инструкции `switch-case`;
- сконструируете кодовый замок.

Цифровая клавиатура

В некоторых сложных проектах, когда плата Arduino не подключена к компьютеру, важно иметь возможность ввода чисел, например секретного кода, чтобы что-то включить или выключить. Для этого можно было бы подключить к цифровым входам десять или более кнопок без фиксации (соответствующих цифрам от 0 до 9). Но намного проще использовать цифровую клавиатуру (рис. 11.1).

Одно из ее преимуществ в том, что она использует всего восемь контактов для 16 кнопок, а для работы с ней есть проверенная библиотека и не нужно добавлять резисторы с целью подтяжки к земле для борьбы с дребезгом контактов, как описывалось в главе 4.

Упомянутую библиотеку с именем Arduino Keypad можно получить по адресу https://github.com/Chris--A/Keypad/archive/master.zip.

Подключение клавиатуры

Подключить цифровую клавиатуру к плате Arduino просто. Поверните клавиатуру лицевой стороной вверх и посмотрите на конец шлейфа. В разъеме вы увидите восемь контактов (рис. 11.2).

Рис. 11.1. Цифровая клавиатура

Рис. 11.2. Штекер для подключения цифровой клавиатуры

Контакты пронумерованы от 8 до 1 слева направо. Во всех проектах с цифровой клавиатурой в этой книге мы будем подключать ее в соответствии с табл. 11.1.

Таблица 11.1. Подключение цифровой клавиатуры к плате Arduino

Номер контакта на цифровой клавиатуре	Номер контакта на плате Arduino
8	Цифровой вход 9
7	Цифровой вход 8
6	Цифровой вход 7
5	Цифровой вход 6
4	Цифровой вход 5
3	Цифровой вход 4
2	Цифровой вход 3
1	Цифровой вход 2

Программная обработка клавиатуры

В скетч для проекта, использующего цифровую клавиатуру, нужно вставить несколько строк, добавляющих поддержку клавиатуры, как отмечено в листинге 11.1. Обязательный код начинается со строки ❶ и заканчивается строкой ❺.

Прежде чем двинуться дальше, проверим работоспособность клавиатуры. Для этого введите и загрузите скетч из листинга 9.1.

Листинг 11.1. Скетч, демонстрирующий работу с цифровой клавиатурой

```
❶ // Начало кода настройки цифровой клавиатуры
  #include "Keypad.h"

  const byte ROWS = 4; // Четыре ряда кнопок
  const byte COLS = 4; // по четыре в каждом ряду
❷ char keys[ROWS][COLS] =
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
❸ byte rowPins[ROWS] = {9, 8, 7, 6};
❹ byte colPins[COLS] = {5, 4, 3, 2};

  Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

❺ // Конец кода настройки цифровой клавиатуры

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

```
void loop()
{
  char key = keypad.getKey();

  if (key != NO_KEY)
  {
    Serial.print(key);
  }
}
```

В строке ❷ объявляется переменная-массив типа `char` с символами — буквами, цифрами или знаками, которые могут генерироваться клавиатурой. В этом примере она содержит цифры и знаки. Строки ❸ и ❹ определяют, какие контакты на Arduino будут задействованы. При желании значения в этих строках можно изменить, чтобы подключить клавиатуру по-другому.

Тестирование скетча

После загрузки скетча откройте окно монитора порта и попробуйте понажимать клавиши на клавиатуре. Символы, соответствующие нажимаемым клавишам, должны появиться в окне монитора порта (рис. 11.3).

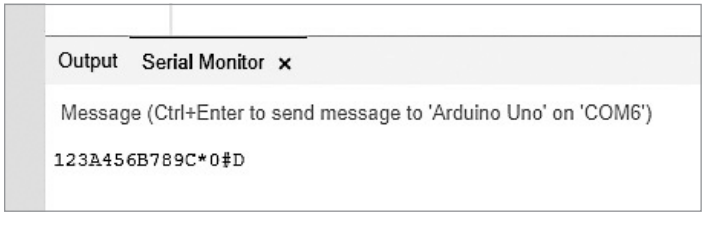


Рис. 11.3. Результаты нажатий клавиш на цифровой клавиатуре

Принятие решений с помощью switch-case

Сравнить переменную с двумя или более значениями проще всего с помощью инструкций `switch-case` вместо `if-then`. Инструкция `switch-case` способна производить произвольное количество сравнений и выполнять код, соответствующий обнаруженному совпадению. Например, если целочисленная переменная `xx` может принимать значения `1`, `2` или `3` и в зависимости от конкретного значения нужно выполнить определенный фрагмент кода. Это можно реализовать, как показано ниже, а не использовать несколько инструкций `if-then`:

```
switch(xx)
{
  case 1:
```

```
    // Выполнить операции, если xx имеет значение 1
    break; // Завершить и продолжить выполнение скетча со строки,
           // следующей за инструкцией switch-case
  case 2:
    // Выполнить операции, если xx имеет значение 2
    break;
  case 3:
    // Выполнить операции, если xx имеет значение 3
    break;
  default:
    // Выполнить операции, если xx имеет какое-то другое значение,
    // отличное от 1, 2 и 3;
    // инструкцию default можно опустить
}
```

Необязательный раздел `default:` в конце этого фрагмента позволяет выполнить некоторые операции при отсутствии совпадений со значениями, указанными выше в инструкции `switch-case`.

Проект 32: кодовый замок

В этом проекте мы создадим программную часть кодового замка, управляемого цифровой клавиатурой. Для этого воспользуемся настройками скетча из листинга 11.1 и добавим обработку шестизначного секретного кода, который надо ввести с цифровой клавиатуры. С помощью монитора последовательного порта мы будем сообщать пользователю, верный ли введенный им код.

Секретный код хранится в скетче, но никогда не отображается. В зависимости от результатов проверки введенного кода скетч будет вызывать разные функции. Чтобы активировать или деактивировать замок, пользователь должен нажать *, а потом ввести код и нажать #.

Скетч

Введите и загрузите следующий скетч:

```
// Проект 32 — кодовый замок

// Начало кода настройки цифровой клавиатуры

#include "Keypad.h"

const byte ROWS = 4; // Четыре ряда кнопок
const byte COLS = 4; // по четыре в каждом ряду
char keys[ROWS][COLS] =
  {'1','2','3','A'},
  {'4','5','6','B'},
```

```
    {'7','8','9','C'},
    {'*','0','#','D'}
  byte rowPins[ROWS] = {9, 8, 7, 6};
  byte colPins[COLS] = {5, 4, 3, 2};

  Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

  // Конец кода настройки цифровой клавиатуры

❶ char PIN[6]={'1','2','3','4','5','6'}; // Секретный код
  char attempt[6]={0,0,0,0,0,0};
  int z=0;

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

  void correctPIN() // Вызывается, если введен верный код
  {
    Serial.println("Correct PIN entered...");
  }

  void incorrectPIN() // Вызывается, если введен неверный код
  {
    Serial.println("Incorrect PIN entered!");
  }

  void checkPIN()
  {
    int correct=0;
❷   for ( int i = 0; i < 6 ; i++ )
    {
      // Выполняет обход элементов массива с шестью символами
      // Если очередной символ совпал с соответствующим символом в PIN,
      // то увеличивается значение счетчика
      if (attempt[i]==PIN[i])
      {
        correct++;
      }
    }
    if (correct==6)
    {
❸     correctPIN();
    }
    else
    {
❹     incorrectPIN();
    }
```

```
    for (int zz=0; zz<6; zz++) // Удалить код, введенный при предыдущей попытке
    {
      attempt[zz]=0;
    }
  }

  void readKeypad()
  {
    char key = keypad.getKey();
    if (key != NO_KEY)
    {
❺     switch(key)
      {
        case '*':
          z=0;
          break;
        case '#':
          delay(100); // Защита от дребезга контактов
          checkPIN();
          break;
        default:
          attempt[z]=key;
          z++;
      }
    }
  }

  void loop()
  {
❻   readKeypad();
  }
```

Принцип действия

После обычной процедуры настройки (из листинга 11.1) скетч начинает непрерывно «сканировать» клавиатуру, вызывая функцию `readKeypad()` ❻. Обнаружив факт нажатия клавиши, функция исследует ее с помощью структуры `switch-case` ❺. Числовые значения, соответствующие цифровым клавишам, записываются в массив `attempt`. Когда пользователь нажимает #, вызывается функция `checkPin()`.

В ❷ значение нажатой клавиши сравнивается с секретным кодом в массиве `PIN` ❶. После введения верной последовательности цифр вызывается функция `correctPin()` ❸. Туда можно добавить свой код, например управляющий замком. Если введена неверная последовательность, вызывается функция `incorrectPin()` ❹. После проверки ввода пользователя массив `attempt` очищается и подготавливается к следующей попытке ввода.

Тестирование скетча

Загрузив скетч в плату Arduino, откройте окно монитора последовательного порта, нажмите клавишу со звездочкой (*) на цифровой клавиатуре, введите секретный код и затем нажмите клавишу с решеткой (#). Попробуйте ввести верный и неверный код. Вы должны увидеть результаты как на рис. 11.4.

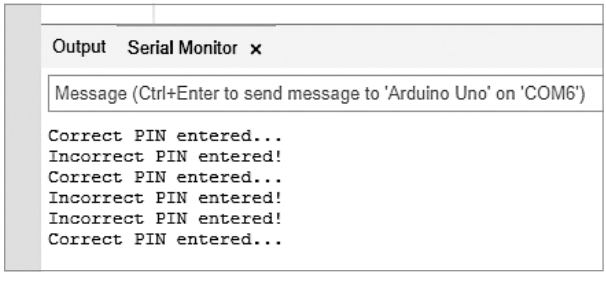


Рис. 11.4. Результаты ввода верного и неверного кода

Что дальше?

В этой главе вы познакомились еще с одним способом ввода информации в плату Arduino. Вы знаете, как использовать цифровую клавиатуру в скетче, и знакомы с возможностями ограничения доступа ко всему, чем может управлять Arduino. Попутно вы встретились с очень полезной инструкцией `switch-case`. В следующей главе мы поговорим еще об одной форме ввода — сенсорном экране.

12

Сенсорные экраны

В этой главе вы:

- узнаете, как подключить резистивный сенсорный экран к плате Arduino;
- научитесь читать значения, которые может возвращать сенсорный экран;
- сконструируете простой выключатель, срабатывающий от прикосновения;
- познакомитесь с функцией `map()`;
- сконструируете выключатель с функцией регулировки света.

Сенсорные экраны повсюду: смартфоны, планшеты и карманные игровые устройства. Так почему бы и нам не использовать их для взаимодействий с пользователем?

Сенсорные экраны

Сенсорные экраны могут стоить очень дорого, но мы будем использовать недорогую модель, доступную в интернет-магазине Adafruit (артикул 333 и 3575), первоначально разработанную для игровой консоли Nintendo DS.

Этот сенсорный экран имеет размеры 6,22 × 7,62 см и изображен на рис. 12.1.

Обратите внимание, что экран соединен шлейфом с маленькой печатной платой справа. Этот *адаптер* используется для соединения сенсорного экрана с макетной платой и Arduino. В комплект входит колодка контактов, которую нужно припаять к плате адаптера. На рис. 12.2 показано увеличенное изображение адаптера.

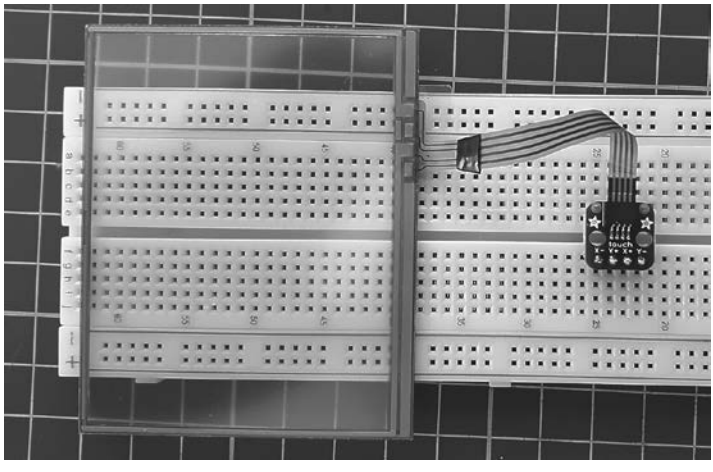


Рис. 12.1. Сенсорный экран, смонтированный на макетной плате

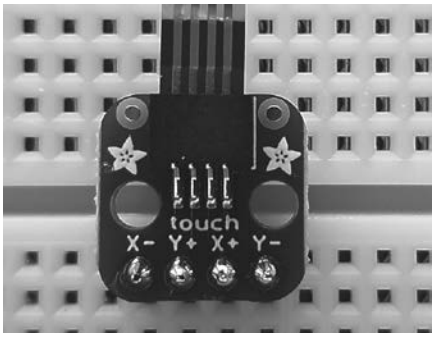


Рис. 12.2. Адаптер сенсорного экрана

Подключение сенсорного экрана

Адаптер сенсорного экрана подключается к плате Arduino, как показано в табл. 12.1.

Таблица 12.1. Подключение адаптера сенсорного экрана к плате Arduino

Контакт на адаптере	Контакт на плате Arduino
X–	A3
Y+	A2
X+	A1
Y–	A0

Проект 33: определение области касания на сенсорном экране

Резистивный сенсорный экран состоит из стеклянной панели и гибкой пластиковой мембраны, между которыми два слоя резистивного покрытия. Одно резистивное покрытие действует как ось *X*, а другое — как ось *Y*. Сопротивление резистивного покрытия изменяется в зависимости от точки касания. То есть, измеряя напряжение на каждом слое, можно определять координаты *x* и *y* области касания.

В этом проекте мы с помощью платы Arduino измерим напряжение в точке касания и преобразуем его в целочисленные координаты.

Оборудование

Ниже перечислено оборудование, которое понадобится для этого проекта:

- сенсорный экран Adafruit (артикул 333);
- один адаптер Adafreuit (артикул 3575);
- несколько отрезков провода разной длины со штекерами «папа-папа» («штекер-штекер»);
- одна макетная плата;
- плата Arduino и кабель USB.

Подключите сенсорный экран к плате Arduino в соответствии с табл. 12.1, а саму плату присоедините к компьютеру кабелем USB.

Скетч

Введите и загрузите следующий скетч:

```
// Проект 33 – определение области касания на сенсорном экране

int x,y = 0;

❶ int readX() // Возвращает координату x на сенсорном экране
{
  int xr=0;
  pinMode(A0, INPUT);
  pinMode(A1, OUTPUT);
  pinMode(A2, INPUT);
  pinMode(A3, OUTPUT);
  digitalWrite(A1, LOW);  // Подать низкий уровень на A1
  digitalWrite(A3, HIGH); // Подать высокий уровень на A3
  delay(5);
  xr=analogRead(0);       // Получить координату x
  return xr;
}
```

```
❷ int readY() // Возвращает координату y на сенсорном экране
  {
    int yr=0;
    pinMode(A0, OUTPUT);    // A0
    pinMode(A1, INPUT);     // A1
    pinMode(A2, OUTPUT);    // A2
    pinMode(A3, INPUT);     // A3
    digitalWrite(14, LOW);  // Подать низкий уровень на A0
    digitalWrite(16, HIGH); // Подать высокий уровень на A2
    delay(5);
    yr=analogRead(1);       // Получить координату y
    return yr;
  }

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

  void loop()
  {
    Serial.print(" x = ");
    x=readX();
❸   Serial.print(x);
    y=readY();
    Serial.print(" y = ");
❹   Serial.println(y);
    delay (200);
}
```

Функции `readX()` и `readY()` (❶ и ❷) измеряют напряжение на резистивных покрытиях сенсорного экрана с помощью `analogRead()` и возвращают полученное значение. Скетч постоянно вызывает эти две функции для определения координат области касания сенсорного экрана в реальном времени и их вывода в монитор порта (❸ и ❹) (задержка `delay(5)` в каждой функции необходима — она дает время аналоговым входам/выходам изменить свое состояние).

Тестирование скетча

Тестируя скетч, понаблюдайте за изменением показаний в окне монитора порта при касании сенсорного экрана. Обратите внимание, как изменяются значения `X` и `Y` в зависимости от позиции точки касания. Посмотрите, какие значения отображаются в окне монитора, когда вы не касаетесь экрана (рис. 12.3).

Запомните эти значения — они пригодятся вам, если в скетче вы будете определять факт отсутствия прикосновения к экрану. Помните, что разные экраны немного

отличаются друг от друга, поэтому лучше составить карту показаний для своей модели и определить границы, в которых она обнаруживает касания.

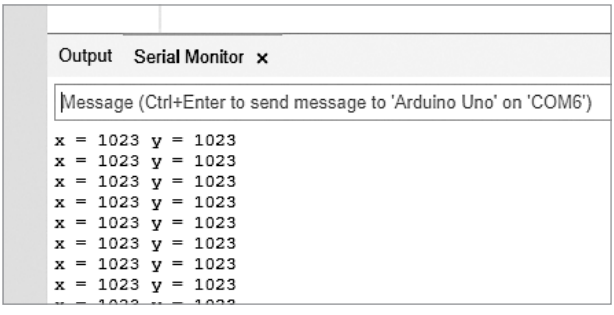


Рис. 12.3. Значения, которые появляются в отсутствие прикосновений к сенсорному экрану

Калибровка сенсорного экрана

Коснувшись сенсорного экрана в углах (рис. 12.4) и записав полученные значения, вы фактически откалибруете его.

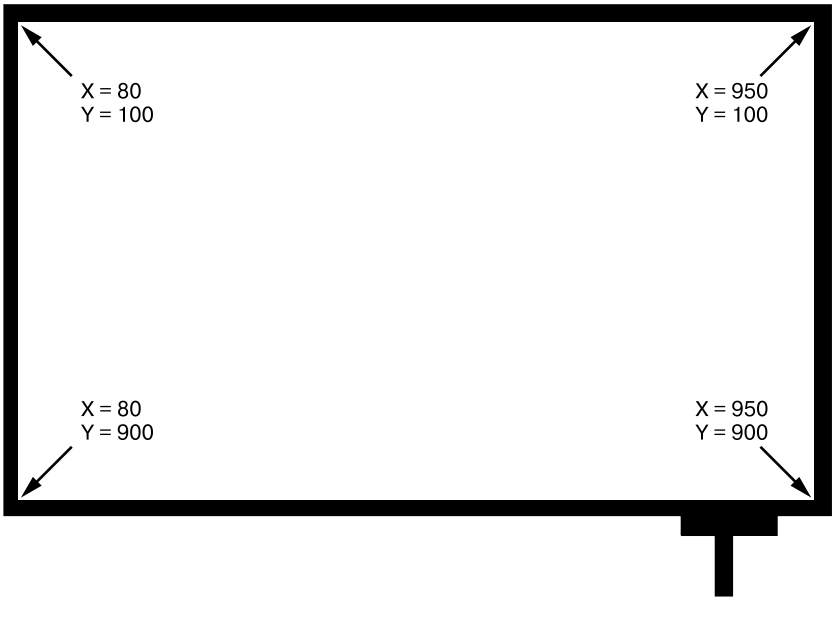


Рис. 12.4. Калибровка сенсорного экрана

Откалибровав сенсорный экран, вы можете математически поделить его на небольшие области, а с помощью функций `readX()` и `readY()` сможете определять факт касания разных управляющих областей на экране и использовать инструкции `if-then` для выполнения определенных операций, как показано в проекте 34.

Проект 34: двухзонный выключатель

В этом проекте мы создадим на основе сенсорного экрана простой выключатель. Для начала поделим экран на две вертикальные зоны (рис. 12.5).

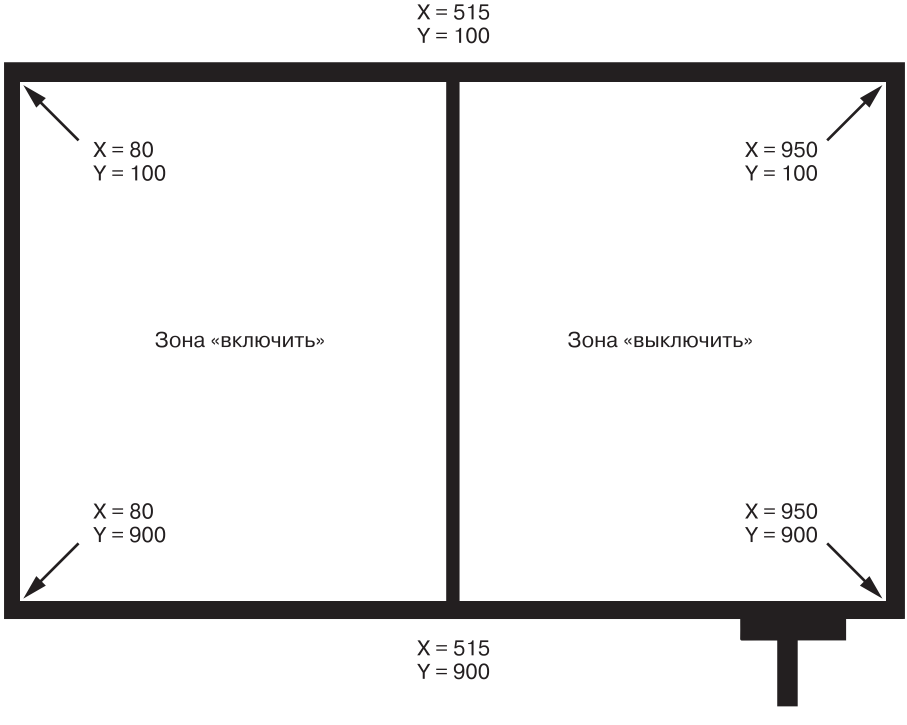


Рис. 12.5. Разметка двухзонного выключателя

Сравнивая координаты точки касания с границами зон, плата Arduino будет определять зону, где произошло касание. После этого код будет устанавливать высокий или низкий уровень напряжения на цифровом выходе 10 (что можно использовать для включения и выключения какого-нибудь устройства).

Скетч

Введите и загрузите следующий скетч:

```
// Проект 34 – двухзонный выключатель

int x,y = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(10, OUTPUT);
}

void switchOn()
{
  digitalWrite(10, HIGH);
  Serial.print("Turned ON at X = ");
  Serial.print(x);
  Serial.print(" Y = ");
  Serial.println(y);
  delay(200);
}

void switchOff()
{
  digitalWrite(10, LOW);
  Serial.print("Turned OFF at X = ");
  Serial.print(x);
  Serial.print(" Y = ");
  Serial.println(y);
  delay(200);
}

int readX() // Возвращает координату x на сенсорном экране
{
  int xr=0;
  pinMode(A0, INPUT);
  pinMode(A1, OUTPUT);
  pinMode(A2, INPUT);
  pinMode(A3, OUTPUT);
  digitalWrite(A1, LOW);  // Подать низкий уровень на A1
  digitalWrite(A3, HIGH); // Подать высокий уровень на A3
  delay(5);
  xr=analogRead(0);
  return xr;
}

int readY() // Возвращает координату y на сенсорном экране
{
  int yr=0;
```

```
  pinMode(A0, OUTPUT);
  pinMode(A1, INPUT);
  pinMode(A2, OUTPUT);
  pinMode(A3, INPUT);
  digitalWrite(A0, LOW);  // Подать низкий уровень на A0
  digitalWrite(A2, HIGH); // Подать высокий уровень на A2
  delay(5);
  yr=analogRead(1);
  return yr;
}

void loop()
{
  x=readX();
  y=readY();
❶ // Касание в зоне "включить"?
  if (x<=515 && x>=80)
  {
    switchOn();
  }
❷ // Касание в зоне "выключить"?
  if (x<950 && x>=516)
  {
    switchOff();
  }
}
```

Принцип действия

Две инструкции `if` в функции `void loop()` определяют зону касания. Если точка касания находится в левой зоне, оно интерпретируется как команда «включить» ❶. Если точка касания в правой зоне, оно интерпретируется как «выключить» ❷.

ПРИМЕЧАНИЕ

Координата y в этом проекте игнорируется, потому что сенсорный экран условно разбит вертикальной границей на две зоны по горизонтали. Если бы мы определили еще и горизонтальные границы, тогда необходимо было бы проверять и координату y, как мы увидим это в проекте 35.

Тестирование скетча

На рис. 12.6 показаны результаты, полученные в процессе тестирования. Для каждого касания выводится состояние выключателя и координаты.

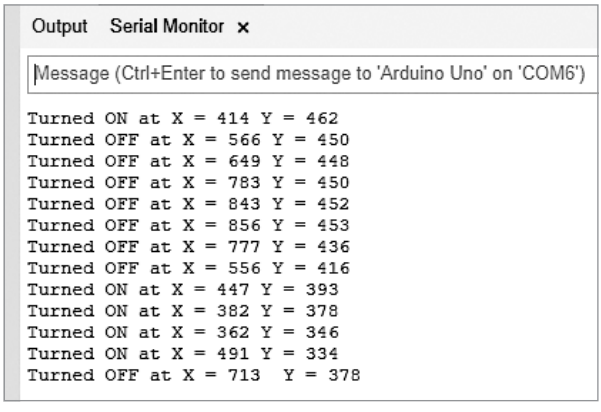


Рис. 12.6. Вывод скетча из проекта 34

Функция map()

В какой-то момент вам может понадобиться преобразовать целое число из одного диапазона в значение из другого диапазона. Например, координата *x* точки касания к сенсорному экрану может принимать значения от 100 до 900, но вам может понадобиться преобразовать их в диапазон от 0 до 255 для управления 8-битным выводом.

Для этого используйте функцию `map()`:

```
map(value, fromLow, fromHigh, toLow, toHigh);
```

Например, вот как можно использовать эту функцию, чтобы преобразовать значение 450 координаты *x* точки касания к сенсорному экрану в диапазон 0–255:

```
x = map(450, 100, 900, 0, 255);
```

Этот вызов вернет значение `95`. Функцию `map()` мы будем использовать в проекте 35.

Проект 35: трехзонный выключатель

В этом проекте мы реализуем трехзонный сенсорный выключатель для светодиода, подключенного к цифровому выходу 3, который будет включать, выключать и регулировать яркость светодиода, используя для этого ШИМ (глава 3).

Разметка сенсорного экрана

На рис. 12.7 показана разметка сенсорного экрана для этого проекта.

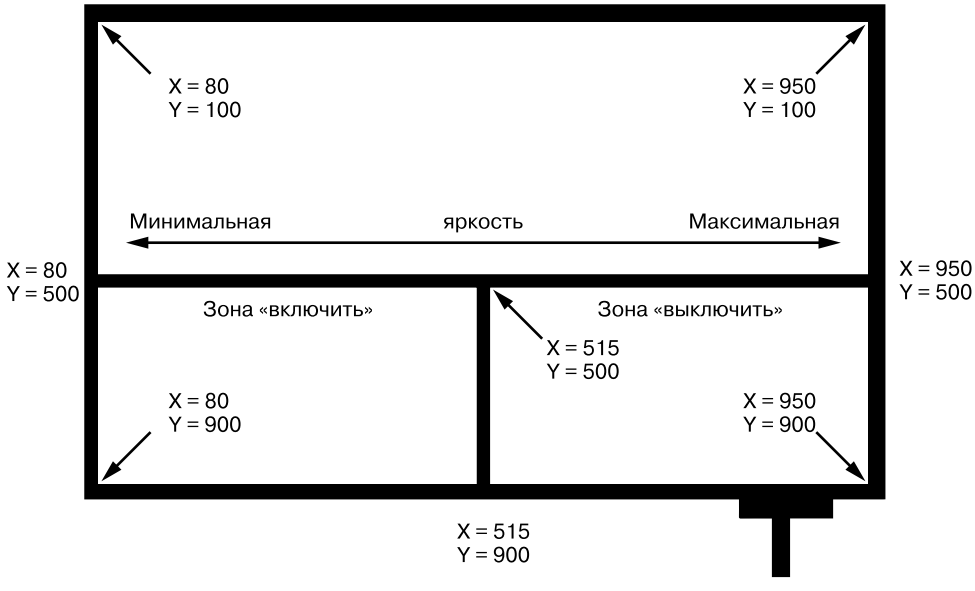


Рис. 12.7. Разметка сенсорного экрана для проекта 35

Сенсорный экран поделен на три зоны: «включить», «выключить» и зона регулировки яркости. По значениям, возвращаемым сенсорным экраном, мы определим, в какой области произошло касание, и выполним соответствующие действия.

Скетч

Введите и загрузите следующий скетч:

```
// Проект 35 – трехзонный выключатель

int x,y = 0;

void setup()
{
  pinMode(3, OUTPUT);
  Serial.begin(9600);
}

void switchOn()
```

```
{
  digitalWrite(3, HIGH);
  delay(200);
}

void switchOff()
{
  digitalWrite(3, LOW);
  delay(200);
}

void setBrightness()
{
  int PWMvalue;
❶ PWMvalue=map(x, 80, 950, 0, 255);
  analogWrite(3, PWMvalue);
}

int readX() // Возвращает координату x на сенсорном экране
{
  int xr=0;
  pinMode(A0, INPUT);
  pinMode(A1, OUTPUT);
  pinMode(A2, INPUT);
  pinMode(A3, OUTPUT);
  digitalWrite(A1, LOW);  // Подать низкий уровень на A1
  digitalWrite(A3, HIGH); // Подать высокий уровень на A3
  delay(5);
  xr=analogRead(0);
  return xr;
}

int readY() // Возвращает координату y на сенсорном экране
{
  int yr=0;
  pinMode(A0, OUTPUT); // A0
  pinMode(A1, INPUT);  // A1
  pinMode(A2, OUTPUT); // A2
  pinMode(A3, INPUT);  // A3
  digitalWrite(A0, LOW);  // Подать низкий уровень на A0
  digitalWrite(A2, HIGH); // Подать высокий уровень на A2
  delay(5);
  yr=analogRead(1);
  return yr;
}

void loop()
{
  x=readX();
  y=readY();
```

```
    // Касание в зоне "включить"?
❷   if (x<=950 && x>=515 && y>= 500 && y<900)
    {
      switchOn();
    }

    // Касание в зоне "выключить"?
❸   if (x>500 && x<=900 && y>= 500 && y<900)
    {
      switchOff();
    }

    // Касание в зоне "яркость"?
❹   if (y>=100 && y<=500)
    {
      setBrightness();
    }
    Serial.println(x);
  }
```

Принцип действия

Как и скетч двухзонного выключателя, этот определяет, где произошло касание — в области «включить» ❷, «выключить» ❸ или «яркость» ❹. В этом проекте первые две области стали меньше, потому что верхняя половина экрана теперь предназначена для управления яркостью. Если касание произошло в области управления яркостью, координата *x* преобразуется в функции `setBrightness()` в относительное значение ШИМ ❶ с помощью `map()` и яркость светодиода устанавливается в соответствии с этим значением.

Эти же базовые функции можно использовать для самых разных выключателей и регуляторов на основе этого простого и недорогого сенсорного экрана. Кроме того, можно написать свою библиотеку, возвращающую координаты *x* и *y* и управляющую яркостью, что вы сможете использовать в будущих скетчах.

Что дальше?

В этой главе мы узнали, что сенсорные экраны — это еще один способ взаимодействия с пользователем и управления Arduino. Дальше мы сосредоточимся на самой плате Arduino, познакомимся с разными ее модификациями и создадим свою версию Arduino на макетной плате для навесного монтажа.

13

Семейство плат Arduino

В этой главе вы:

- узнаете, как собрать собственную версию Arduino на макетной плате;
- исследуете преимущества и особенности широкого спектра плат, совместимых с Arduino;
- познакомитесь с открытым аппаратным обеспечением.

В этой главе мы исследуем устройство платы Arduino и соберем свою версию на макетной плате. Эти знания помогут сэкономить деньги, особенно тем, кто активно занимается изменением проектов и созданием прототипов. Здесь вы познакомитесь и с новыми компонентами и схемами. Далее вы исследуете способы загрузки скетчей в самодельную плату Arduino без применения дополнительного оборудования. В заключение мы рассмотрим типичные альтернативы плате Arduino Uno и их отличия.

Проект 36: создание собственной платы Arduino

С увеличением количества или сложности проектов затраты на приобретение плат Arduino легко могут выйти из-под контроля. В таком случае проще и дешевле интегрировать схему Arduino в проекты, собирая ее на макетной плате, где можно разместить дополнительные компоненты для реализации конкретного проекта. Стоимость всех составляющих для воспроизведения простейшей версии Arduino на макетной плате (которая обычно допускает многократное использование при аккуратном обращении) не должна превышать 10 долларов США. Если в проекте

нужно использовать много внешних компонентов, проще собрать свою версию Arduino. В этом случае не придется тянуть массу проводов от Arduino к макетной плате.

Оборудование

Для сборки минимальной версии Arduino понадобится следующее оборудование:

- одна макетная плата;
- несколько отрезков провода разной длины;
- один стабилизатор напряжения 7805;
- один кварцевый резонатор частотой 16 МГц;
- один микроконтроллер ATmega328P-PU с загрузчиком Arduino;
- один электролитический конденсатор емкостью 1 мкФ и рабочим напряжением 25 В (C1);
- один электролитический конденсатор емкостью 100 мкФ и рабочим напряжением 25 В (C2);
- два керамических конденсатора емкостью 22 пФ и рабочим напряжением 50 В (C3 и C4);
- один керамический конденсатор емкостью 100 нФ и рабочим напряжением 50 В (C5);
- два резистора номиналом 560 Ом (R1 и R2);
- один резистор номиналом 10 кОм (R3);
- два светодиода по вашему выбору (LED1 и LED2);
- одна кнопка без фиксации (S1);
- одна колодка с шестью контактами;
- один разъем для подключения батареи типа PP3 (6F22/Крона);
- одна батарея типа PP3 (6F22/Крона) напряжением 9 В.

Некоторые компоненты могут быть вам незнакомы. В следующих разделах вы подробнее о них узнаете, увидите их фотографии и их обозначения на принципиальных схемах.

Линейный стабилизатор напряжения 7805

Стабилизатор содержит простую схему, преобразующую одно напряжение в другое. В список необходимого оборудования включен стабилизатор 7805, который может преобразовывать напряжение от 7 до 30 вольт в фиксированное напряжение 5 вольт

с силой тока до 1 А. Этого вполне достаточно для самодельной платы Arduino. На рис. 13.1 можно видеть типичного представителя стабилизаторов из серии 7805 в корпусе TO-220.

На рис. 13.2 показано, как стабилизатор 7805 изображается на принципиальных схемах. Если смотреть со стороны маркировки, левый вывод (IN) обозначен буквой J и служит для подачи входного напряжения, центральный вывод (GND) подключается к «земле» и правый вывод (OUT) — под буквой G — выходное напряжение 5 В. Металлический язычок сверху с круглым отверстием позволяет смонтировать стабилизатор на массивной металлической основе, которую часто называют *теплоотводом*, или *радиатором*. Наличие радиатора обязательно, если ток нагрузки близок к максимальному (1 А). В таком режиме работы стабилизатор 7805 очень сильно греется. Металлический язычок соединен и с выводом GND. В этом проекте нам нужен один стабилизатор напряжения 7805.

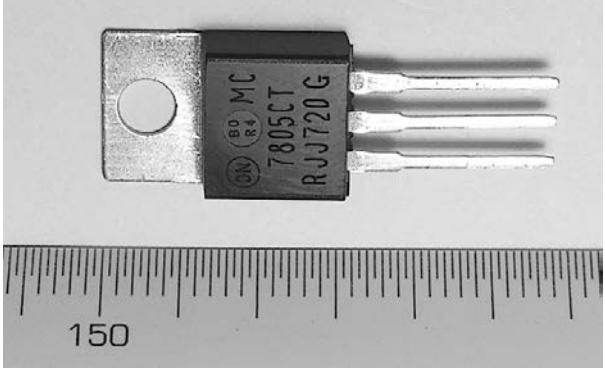


Рис. 13.1. Стабилизатор напряжения 7805

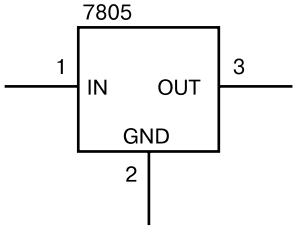


Рис. 13.2. Обозначение стабилизатора напряжения 7805

Кварцевый резонатор частотой 16 МГц

Кварцевый резонатор часто называют просто *кварцем*. Он помогает поддерживать частоту электрических колебаний с очень высокой точностью. В данном случае частота равна 16 МГц. Кварц, используемый в этом проекте, можно увидеть на рис. 13.3.

Сравните это изображение с кварцем на вашей плате Arduino. Они должны быть идентичны по форме и размеру.

Кварц не имеет полярности. На принципиальных схемах он изображается так, как на рис. 13.4.

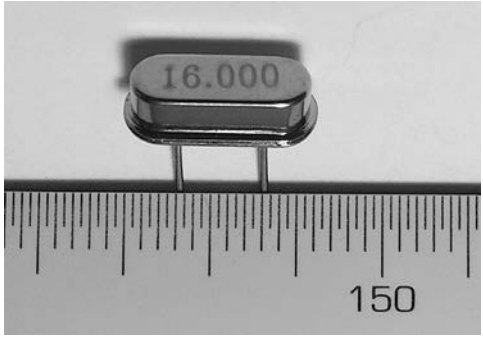


Рис. 13.3. Кварцевый резонатор

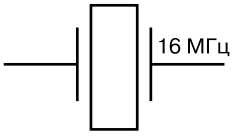


Рис. 13.4. Обозначение кварцевого резонатора

Кварц определяет быстродействие микроконтроллера. Например, микроконтроллер, который мы будем использовать, работает с тактовой частотой 16 МГц. То есть он выполняет 16 миллионов элементарных инструкций в секунду. Но это не значит, что строки в скетче будут выполняться с такой скоростью, потому что каждая строка исходного кода преобразуется во время компиляции в множество элементарных инструкций.

Микроконтроллер Atmel ATmega328P-PU

Как мы знаем из главы 2, *микроконтроллер* — это маленький компьютер, мозг платы Arduino. В нем есть выполняющий инструкции процессор, несколько видов памяти для хранения данных и инструкций скетча. Микроконтроллер поддерживает несколько способов приема и передачи данных. На рис. 13.5 можно увидеть, как он выглядит, на примере модели ATmega328P-PU. Заметьте, что вывод микроконтроллера с номером 1 находится на фотографии слева внизу и отмечен маленькой точкой.

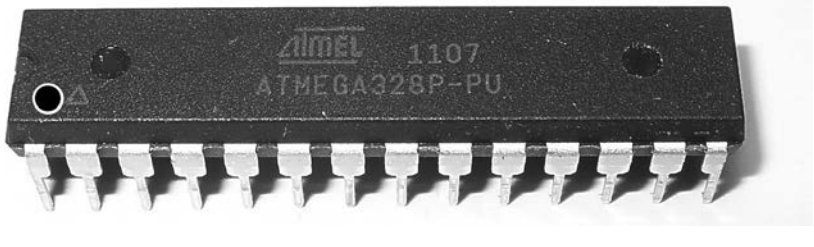


Рис. 13.5. Микроконтроллер ATmega328P-PU

На принципиальных схемах микроконтроллер изображается как на рис. 13.6.

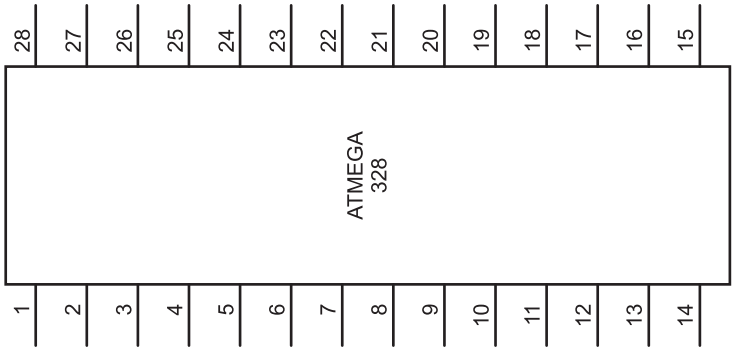


Рис. 13.6. Обозначение микроконтроллера

Не во всех микроконтроллерах есть *загрузчик* Arduino — программа, которая загружает и запускает скетчи для Arduino. Выбирая микроконтроллер для сборки самодельной платы Arduino, проверьте, есть ли в нем загрузчик. Обычно такие микроконтроллеры можно купить там же, где есть платы Arduino, например у Adafruit, PMD Way и SparkFun.

Схема

На рис. 13.7 изображена принципиальная схема нашей версии платы Arduino.

У этой схемы два раздела. Первый слева — это источник питания, который понижает входное напряжение до стабильных 5 В. Когда на плату подается питание, зажигается светодиод LED1. Второй раздел, справа, включает микроконтроллер, кнопку сброса, контакты последовательного интерфейса для программирования и еще один светодиод. Этот светодиод подключен к выводу ATmega328P-PU, который на обычной плате Arduino связан с контактом 13.

Соедините компоненты как на схеме. Не забудьте протянуть провода к колодке с шестью контактами (рис. 13.8), которая внизу на принципиальной схеме изображена как шесть кружочков. Далее мы будем использовать эти контакты для загрузки скетча в самодельную плату Arduino.

Схема питается от простой батареи напряжением 9 В, присоединяемой простым разъемом (рис. 13.9). Подключите красный вывод разъема к положительному (+) контакту, а черный — к отрицательному (–).

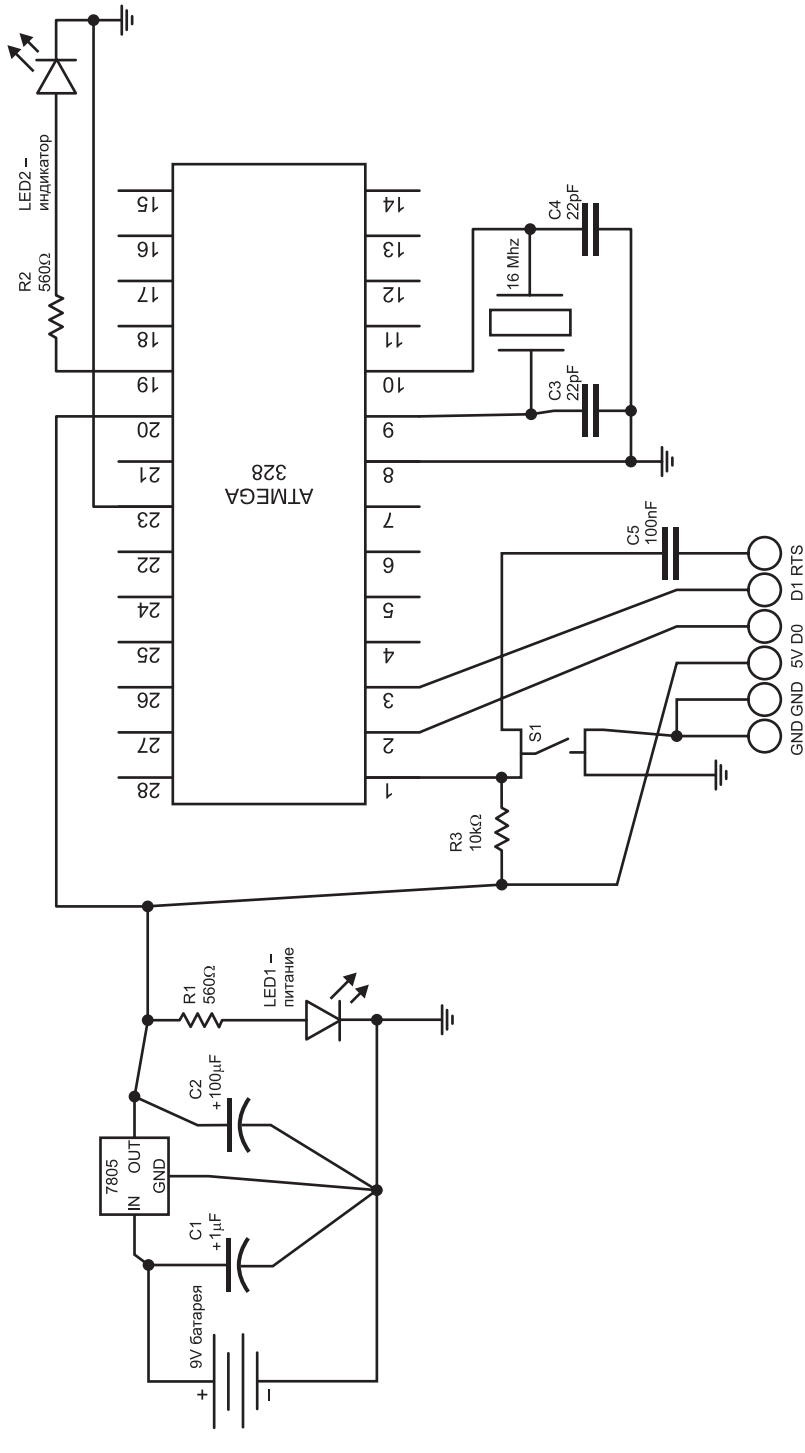


Рис. 13.7. Принципиальная схема минимальной платы Arduino

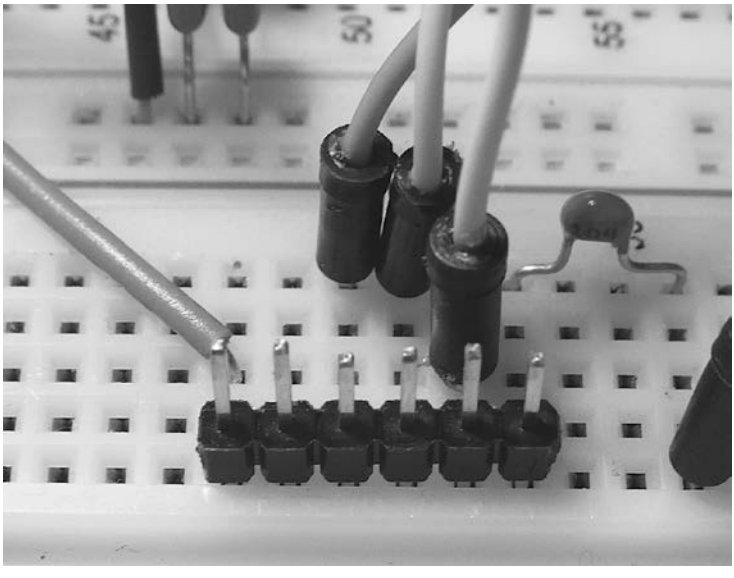


Рис. 13.8. Колодка с шестью контактами

Рис. 13.9. Батарея 9 В и разъем для ее подключения

Идентификация выводов Arduino

Где же на нашей самодельной плате все остальные контакты (аналоговые, цифровые и другие с обычной платы Arduino)? Они есть и на нашей самодельной версии, просто подключайтесь напрямую к выводам микроконтроллера.

Резистор R2 и светодиод LED2 на самодельной плате Arduino подключены к цифровому контакту 13. В табл. 13.1 перечислены контакты на плате Arduino (слева) и соответствующие им выводы микроконтроллера ATmega328P-PU (справа).

Таблица 13.1. Назначение выводов ATmega328P-PU

Контакт на плате Arduino	Вывод микроконтроллера ATmega328P-PU
RST	1
RX/D0	2
TX/D1	3
D2	4
D3	5
D4	6
5 V	7
GND	8
D5	11
D6	12
D7	13
D8	14
D9	15
D10	16
D11	17
D12	18
D13	19
5 V	20
AREF	21
GND	22
A0	23
A1	24
A2	25
A3	26
A4	27
A5	28

Чтобы можно было избежать путаницы, продавцы (например, Adafruit и Freetronics) продают самоклеящиеся этикетки на микроконтроллер, как на рис. 13.10 (https://www.freetronics.com.au/collections/arduino/products/microcontroller-labels-arduino-pinout/).

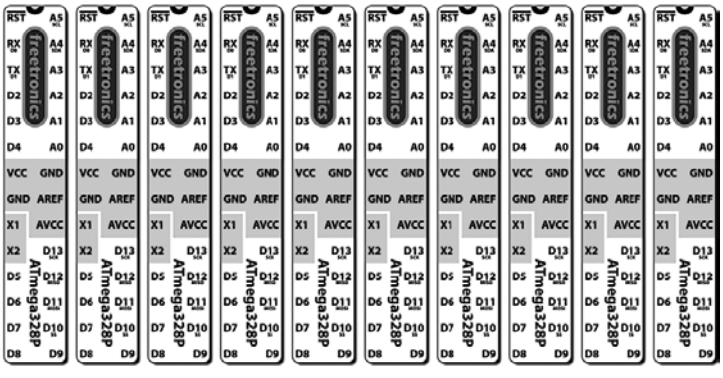


Рис. 13.10. Наклейки для маркировки выводов микроконтроллера

Запуск проверочного скетча

Теперь пришло время загрузить скетч. Для начала загрузим тот, который просто мигает светодиодом:

```
// Проект 36 – создание собственной платы Arduino

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

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Скетч можно загрузить одним из трех способов.

Метод замены микроконтроллера

Самый простой и наименее затратный способ — извлечь микроконтроллер из платы Arduino, вставить микроконтроллер для самодельной платы, загрузить скетч и вернуть запрограммированный микроконтроллер на место.

Для безопасного извлечения микроконтроллера из платы используйте специальный экстрактор (рис. 13.11).

Рис. 13.11. Используйте экстрактор для извлечения микроконтроллера

Извлекайте микроконтроллер аккуратно и не торопясь, чтобы все выводы извлекались из панельки *одновременно*! Это может потребовать определенных усилий, но в конце концов микроконтроллер выйдет из гнезд.

Чтобы вставить его в панельку на плате Arduino, может понадобиться немного подогнуть его выводы, чтобы они точно входили в гнезда. Для этого уприте микроконтроллер одной стороной в плоскую поверхность и осторожно надавите сверху вниз. Повторите процедуру с другой стороной, как показано на рис. 13.12.

Когда будете вставлять микроконтроллер в плату Arduino, не забудьте, что метка на корпусе должна быть справа (рис. 13.13).

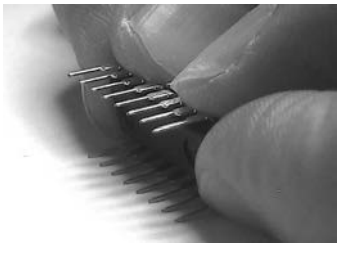


Рис. 13.12. Подгибание выводов микроконтроллера

Рис. 13.13. Правильная ориентация микроконтроллера на плате Arduino

Подключение к имеющейся плате Arduino

Для загрузки скетча в микроконтроллер на самодельной плате можно использовать интерфейс USB на плате Arduino Uno. Это позволяет уменьшить износ панельки для микроконтроллера на плате и сэкономить деньги, ведь не нужно покупать отдельный кабель USB.

Рассмотрим, как загрузить скетч в микроконтроллер с использованием интерфейса USB.

1. Отключите кабель USB от платы Arduino Uno и извлеките из нее микроконтроллер.
2. Выключите питание самодельной платы (если включено).
3. Соедините проводами цифровой контакт 0 на Arduino с выводом 2 микроконтроллера ATmega328P-PU на самодельной плате и цифровой контакт 1 на плате Arduino с выводом 3 микроконтроллера ATmega328P-PU на самодельной плате.
4. Соедините контакты 5 V и GND платы Uno с соответствующими контактами на самодельной плате.
5. Соедините проводом контакт RST на плате Arduino RST с выводом 1 микроконтроллера ATmega328P-PU на самодельной плате.
6. Подключите кабель USB к плате Arduino Uno.

После этого вся система должна действовать, как если бы она состояла из единственной платы Arduino Uno. В результате вы сможете загрузить скетч в микроконтроллер на самодельной плате и использовать монитор последовательного порта при необходимости.

С использованием кабеля FTDI

Последний метод — самый простой, но для него нужен кабель USB *(кабель FTDI)* (такое название объясняется тем, что внутри него есть микросхема интерфейса USB от компании FTDI). Покупайте кабель FTDI, который предназначен для моделей с питанием 5 В[1], потому что кабель для моделей с питанием 3,3 В нам не подходит. Этот кабель (рис. 13.14) имеет разъем USB с одного конца и плоский разъем с шестью контактами — с другого. Разъем USB содержит внутри микросхему,

[1] Будьте осторожны с приобретением кабеля на базе микросхемы FT232. Если вы купите кабель не с оригинальной микросхемой, а с китайским клоном, драйвер Windows от FTDI приведет вашу микросхему в негодность. Старые драйверы (версии 2.8.14 и ранее) работают нормально. С Linux никаких проблем нет. Проблема широко обсуждается в сети. При необходимости микросхему можно восстановить, но с новыми драйверами она работать не будет. — *Примеч. науч. ред.*

эквивалентную микросхеме интерфейса USB на плате Arduino Uno. Плоский разъем с шестью контактами подключается к колодке на плате, изображенной на рис. 13.7 и 13.8.

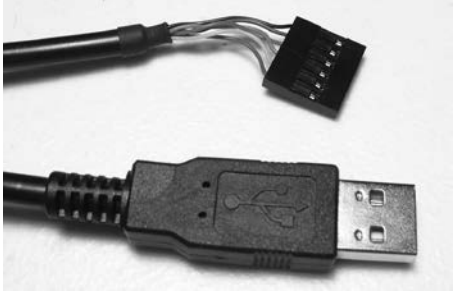


Рис. 13.14. Кабель FTDI

При подключении плоского разъема к колодке на плате черный провод должен соединяться с контактом GND на колодке. После подключения кабеля по нему будет подаваться питание, как и в случае с обычной платой Arduino.

Прежде чем загружать скетч и использовать монитор последовательного порта, измените тип платы на Arduino Duemilanove or Diecimila в меню Tools ▸ Board (Инструменты ▸ Плата), как показано на рис. 13.15.

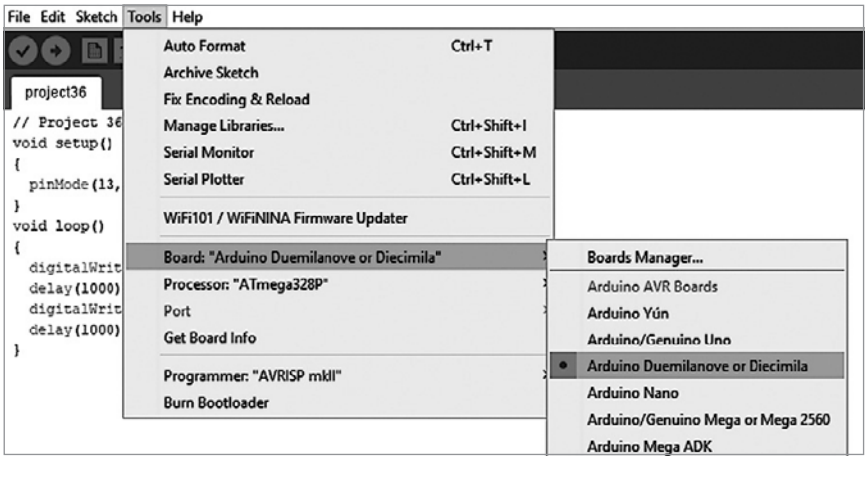


Рис. 13.15. Выбор типа платы в IDE

Выбрав способ по вкусу, проверьте его, попытавшись загрузить скетч из проекта 36. После этого можно приступать к разработке более сложных схем, используя единственную макетную плату, которая позволит вам создать большее количество проектов за меньшие деньги. Вы сможете даже создавать постоянные устройства с нуля, если научитесь самостоятельно делать печатные платы.

Обширное семейство плат Arduino и их заменителей

Во всех проектах из этой книги используется только плата Arduino Uno, но есть огромное множество альтернатив. Они различаются размерами, количеством входов и выходов, объемом памяти для скетчей и ценой.

Одно из существенных различий моделей Arduino — в используемом микроконтроллере. Сейчас наиболее широкое распространение получили микроконтроллеры ATmega328 и ATmega2560, но в модели Due используется другой, более мощный микроконтроллер. Основные различия между ними (включая обе версии ATmega328) перечислены в табл. 13.2.

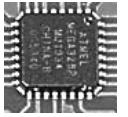
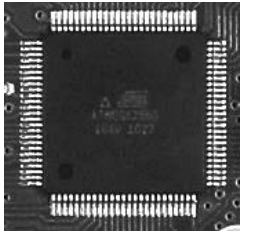
Обычно при сравнении моделей Arduino учитываются типы и объем памяти каждой модели. Всего есть три типа памяти:

- *флеш-память* — предназначена для хранения скетча после его компиляции и загрузки из IDE;
- *EEPROM* (электрически стираемое перепрограммируемое постоянное запоминающее устройство, в данном случае используется как память данных) — небольшой объем памяти для хранения переменных. Вы познакомитесь с ним в главе 19;
- *ОЗУ* — оперативная память для хранения используемых программами переменных.

ПРИМЕЧАНИЕ

Помимо Arduino Uno есть много других моделей, и некоторые описываемые здесь — лишь вершина айсберга. Когда вы приступите к планированию больших или сложных проектов, не бойтесь обращаться к более мощным моделям Mega. Если же вам нужно лишь несколько входных и выходных контактов для постоянного проекта, обратите внимание на модель Nano или даже LilyPad.

Таблица 13.2. Сравнительные характеристики микроконтроллеров

	ATmega328P-PU	ATmega328P SMD	ATmega2560	SAM3X8E
				
Извлекаемый	Да	Нет	Нет	Нет
Тактовая частота	16 МГц	16 МГц	16 МГц	84 МГц
Рабочее напряжение	5 В	5 В	5 В	3,3 В
Количество цифровых контактов	14 (6 с поддержкой ШИМ)	14 (6 с поддержкой ШИМ)	54 (14 с поддержкой ШИМ)	54 (12 с поддержкой ШИМ)
Количество аналоговых контактов	6	8	16	12
Ток на один контакт входа/выхода	40 мА	40 мА	40 мА	От 3 до 15 мА
Доступный объем флеш-памяти	31,5 Кбайт	31,5 Кбайт	248 Кбайт	512 Кбайт
Объем EEPROM	1 Кбайт	1 Кбайт	4 Кбайт	Нет
Объем ОЗУ	2 Кбайт	2 Кбайт	8 Кбайт	96 Кбайт

Теперь исследуем линейку доступных плат.

Arduino Uno

Модель Uno считается стандартом Arduino. Все когда-либо производившиеся платы расширения для Arduino должны быть совместимы с ней. Благодаря встроенному интерфейсу USB и извлекаемому микроконтроллеру плата Arduino Uno проста в использовании.

Freetronics Eleven

На рынке есть много плат, имитирующих работу Arduino Uno. Некоторые даже отличаются улучшенной конструкцией. Одна из таких плат — Freetronics Eleven (рис. 13.16).

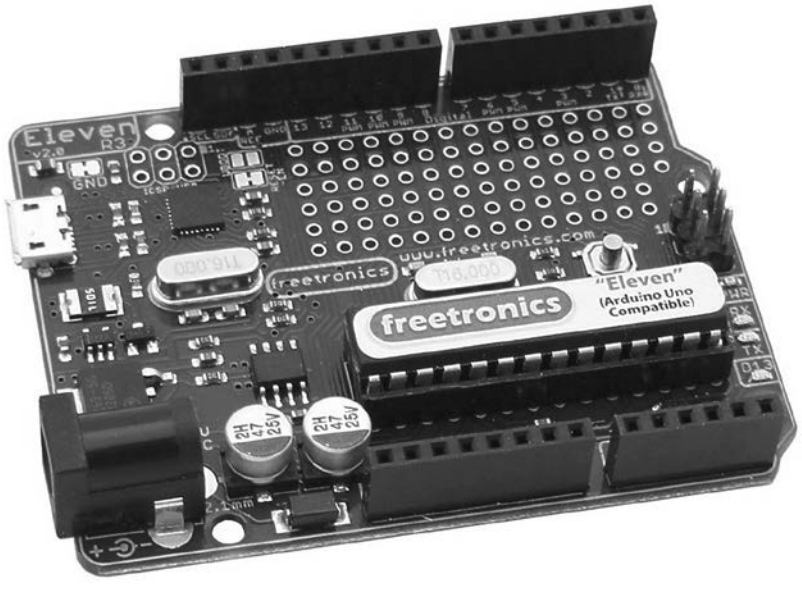


Рис. 13.16. Freetronics Eleven

Несмотря на полную совместимость с Arduino Uno, плата Eleven имеет несколько усовершенствований, придающих ей особую привлекательность. Первое — наличие достаточно большой площадки для макетирования ниже ряда контактов цифровых входов/выходов. Это позволяет собирать собственные схемы прямо на основной плате. Такой подход дает возможность сэкономить деньги и пространство, потому что не приходится покупать отдельную макетную плату.

Второе усовершенствование: контакты для подключения линии приема/передачи данных (TX/RX), питания и встроенный светодиод, подключенный к контакту D13, размещены близко к правому краю платы. Такая компоновка обеспечивает их видимость даже при подключенной плате расширения. Наконец, на ней есть разъем микро-USB, который намного меньше стандартного, устанавливаемого на модели Uno. Это упрощает проектирование компоновки плат расширения — меньше вероятность замкнуть проводники на корпус разъема USB. Модель доступна по адресу http://www.freetronics.com/products/eleven/.

Adafruit Pro Trinket

Плата Adafruit Pro Trinket (рис. 13.17) — это миниатюрная версия Arduino Uno. Она предназначена для создания карманных электронных устройств, использования с макетными платами и в проектах, где необходима плата маленьких размеров.

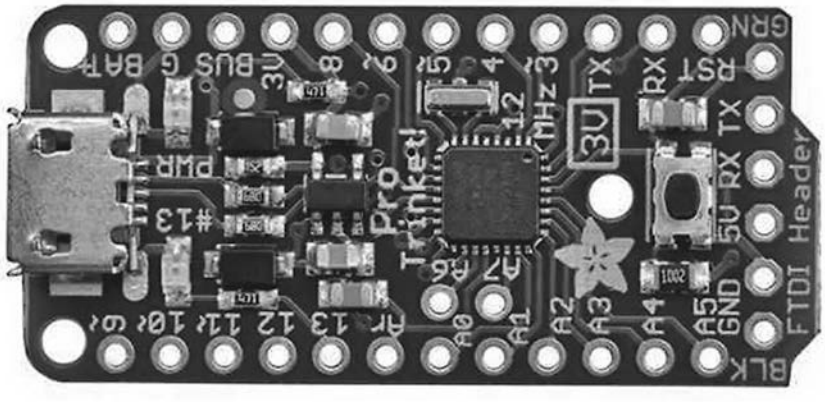


Рис. 13.17. Adafruit Pro Trinket

Она немного отличается от Arduino Uno (например, нет возможности связи по последовательному интерфейсу, кроме как через внешний кабель FTDI). Но невысокая цена делает эту плату очень привлекательной. Приобрести Pro Trinket можно по адресу http://www.adafruit.com/trinket/.

Arduino Nano

Если компактность играет решающую роль, отличным выбором может стать готовая Arduino-совместимая плата Nano. Эта маленькая, но мощная плата Arduino (рис. 13.18) предназначена для использования вместе с макетными платами.

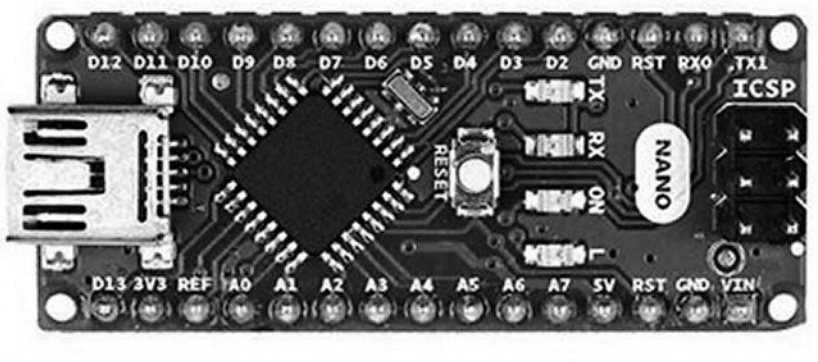


Рис. 13.18. Arduino Nano

Плата Nano имеет размеры 1,8 × 4,3 см, но обладает всеми функциональными возможностями классической Arduino Duemilanove. Кроме того, в ней используется микроконтроллер ATmega328P в планарном корпусе, имеющий два дополнительных контакта аналоговых выходов (A6 и A7). Приобрести Nano можно по адресу https://store.arduino.cc/usa/arduino-nano/.

Arduino LilyPad

Модель LilyPad разрабатывалась для встраивания в нестандартные проекты (карманные электронные устройства). Она не боится влаги и мягкодействующих моющих средств. Это делает LilyPad идеальной, например, для управления светящимися элементами одежды. На рис. 13.19 можно увидеть уникальную конструкцию платы.

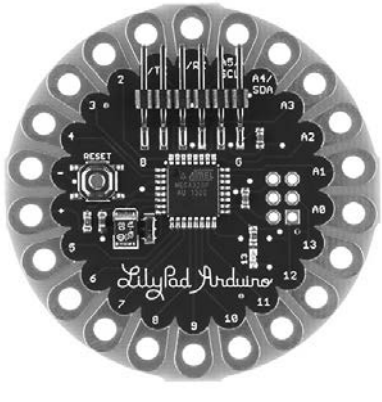


Рис. 13.19. Arduino LilyPad

Контакты на LilyPad выполнены в виде площадок, поэтому провода к ним должны припаиваться. Из-за этого плата лучше подходит для создания постоянных, неразборных устройств. Из-за такого минимализма на плате нет стабилизатора напряжения, поэтому пользователь должен обеспечить источник питания с напряжением от 2,7 до 5,5 В. На плате LilyPad нет и интерфейса USB, поэтому для загрузки скетчей нужно применять 5-вольтовый кабель FTDI. Купить плату Arduino LilyPad можно у любого продавца Arduino.

Arduino Mega 2560

Если контактов входов/выходов на стандартной Arduino Uno недостаточно или нужен больший объем памяти для скетчей, обратите внимание на модель Mega 2560 (рис. 13.20). Она гораздо больше, чем Arduino, — 11 × 5,3 см.

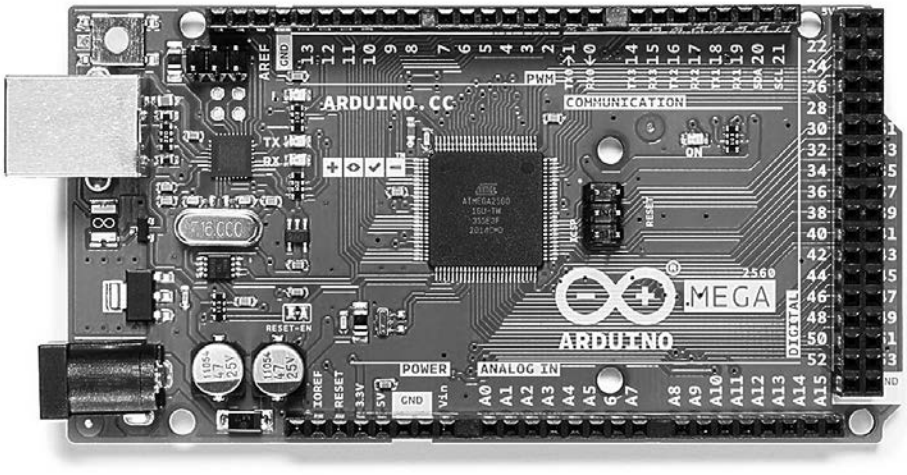


Рис. 13.20. Arduino Mega 2560

Несмотря на размер Mega 2560, она допускает возможность подключения к ней большинства плат расширения для Arduino. Для больших проектов выпускаются макетные платы расширения с размерами, соответствующими размерам модели Mega. В плате Mega используется микроконтроллер ATmega2560 с бóльшим объемом памяти и количеством входов/выходов (как описывается в табл. 13.2) в сравнении с Uno. Кроме того, наличие четырех отдельных линий последовательного интерфейса расширяет ее возможности приема и передачи данных. Приобрести плату Mega 2560 можно у любого продавца Arduino.

Freetronics EtherMega

Если для проекта вам нужна плата Arduino Mega 2560, плата расширения с картой памяти microSD и плата расширения с интерфейсом Ethernet для подключения к интернету, лучшей альтернативой может стать модель EtherMega (рис. 13.21). Она обладает всеми этими функциями, но обойдется дешевле, чем перечисленные три компонента. Приобрести плату EtherMega можно по адресу http://www.freetronics.com/em/.

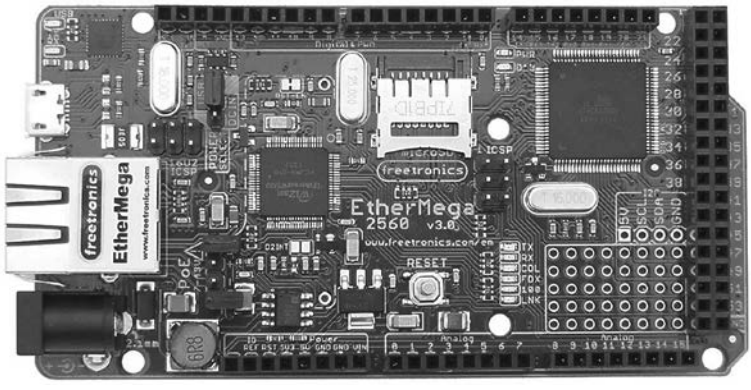


Рис. 13.21. Freetronics EtherMega

Arduino Due

Процессор этой платы с тактовой частотой 84 МГц способен выполнять скетчи намного быстрее. На рис. 13.22 можно увидеть, что эта плата очень похожа на модель Arduino Mega 2560. Но в отличие от последней у Arduino Due есть дополнительный порт USB для подключения внешних устройств и другая маркировка контактов.

Кроме того, объем памяти Due в 16 раз превышает объем памяти на плате Uno. Это позволяет создавать по-настоящему сложные скетчи. Но Due использует рабочее напряжение всего 3,3 В, поэтому любые конструкции, платы расширения и другие устройства, подключаемые к аналоговым или цифровым контактам, не должны работать от напряжения выше этой отметки. Несмотря на эти ограничения, преимущества Due часто оправдывают необходимость внесения изменений в аппаратную часть проектов.

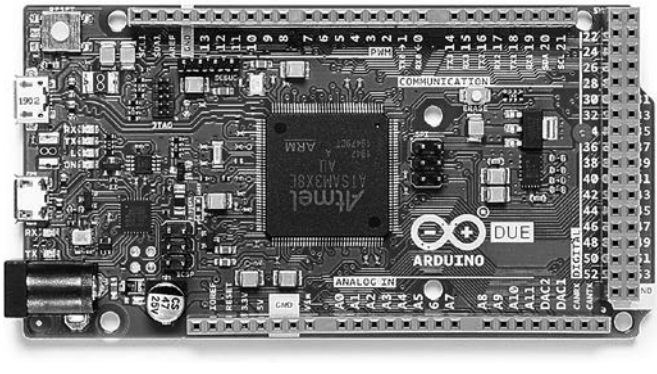


Рис. 13.22. Arduino Due

ПРИМЕЧАНИЕ

Приобретая очередную плату Arduino или аксессуары, обращайтесь к продавцу с хорошей репутацией, который предоставляет поддержку и гарантии. Интернет переполнен предложениями недорогих вариантов, но их производители часто упрощают конструкцию для максимального уменьшения цены, и у вас может не оказаться возможности вернуть неисправный или не отвечающий требованиям товар.

ОТКРЫТОЕ АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Плата Arduino имеет открытую конструкцию, описание которой есть в общем доступе. Поэтому любой желающий может производить, изменять, распространять и использовать эти платы как посчитает нужным. Все это подпадает под определение «открытое аппаратное обеспечение» — недавно появившееся движение, созданное в противовес идеям защиты авторских прав и интеллектуальной собственности. Разработчики Arduino решили раскрыть свою конструкцию, чтобы способствовать развитию сообщества любителей электроники.

Следуя тому же принципу, многие производители аксессуаров или модифицированных версий плат Arduino публикуют конструкторскую документацию для своих устройств под той же открытой лицензией. Это обеспечивает более быстрое развитие продуктов, чем могла бы обеспечить организация, занимающаяся разработкой в одиночестве.

Что дальше?

В этой главе вы узнали о линейке доступных устройств, включая собранную на макетной плате самодельную плату Arduino, познакомились с составляющими Arduino и узнали, как собрать свою плату Arduino на макетной плате для навесного монтажа. Теперь вы можете создать несколько прототипов на основе Arduino и не покупать много плат. Вы познакомились и с разнообразием моделей Arduino на рынке и теперь сможете выбрать ту, что полностью соответствует вашим потребностям. Наконец, вы узнали о существовании движения за открытое аппаратное обеспечение.

В следующей главе вы научитесь пользоваться разными электродвигателями и приступите к созданию своего робота с электромотором, управляемого платой Arduino!

14

Электродвигатели и движение

В этой главе вы:

- узнаете, как использовать сервомотор для создания аналогового термометра;
- научитесь управлять небольшими шаговыми моторами;
- познакомитесь с платой расширения для Arduino, управляющей электродвигателями;
- приступите к созданию робота с электродвигателем;
- узнаете, как с помощью микровыключателей избегать столкновений;
- освоите применение инфракрасных и ультразвуковых датчиков для предотвращения столкновений.

Реализация небольших перемещений с помощью сервомоторов

Серво (сервомотор) состоит из электродвигателя, который по команде может выполнять поворот вала на определенный угол. Соединив вал сервопривода с другими механизмами — колесами, шестернями и рычагами, — вы можете точно управлять объектами во внешнем мире. Сервопривод можно использовать, например, для дистанционного управления автомобилем, если соединить его с помощью *качалки* с рулевым колесом. Примером такого рычага может служить стрелка аналоговых часов. На рис. 14.1 изображен сервопривод с тремя типами качалок.

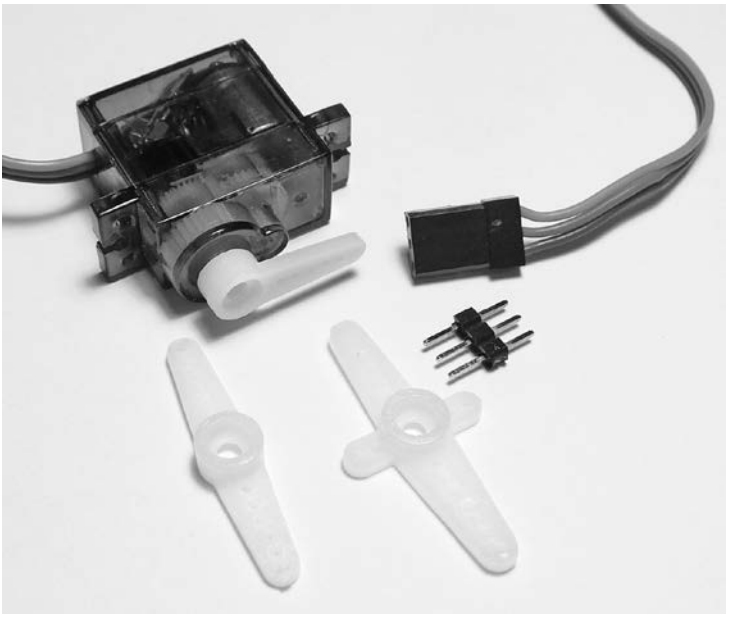


Рис. 14.1. Сервопривод и несколько видов качалок

Выбор серво

При выборе сервопривода учитывайте следующие параметры:

- **скорость** — время, необходимое серво для поворота вала. Обычно измеряется в секундах на угловой градус, оборотах в минуту или в секундах поворота на 60 градусов;
- **диапазон поворота** — угол поворота сервопривода. Например, 180 (половина полного оборота) или 360 градусов (один полный оборот);
- **ток** — сила тока, потребляемого серво. При организации управления сервоприводами с помощью Arduino может понадобиться внешний источник для питания сервопривода;
- **крутящий момент** — усилие, которое способен развить серво. Чем больше крутящий момент, тем более тяжелыми конструкциями сможет управлять серво. Вообще, крутящий момент прямо пропорционален силе потребляемого тока.

На рис. 14.1 изображен сервопривод SG90. Это недорогой экземпляр с диапазоном поворота 180 градусов (рис. 14.2).

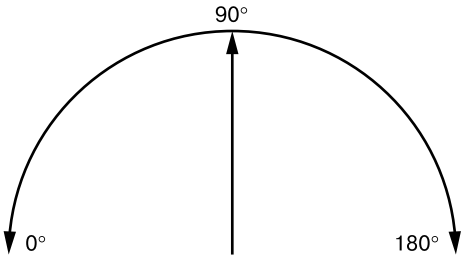


Рис. 14.2. Пример диапазона поворота сервопривода

Подключение сервопривода

Подключить сервопривод к плате Arduino просто, потому что у него всего три вывода. Если вы используете SG90, то самый темный провод (см. рис. 14.1) должен подключаться к контакту GND, центральный — к напряжению питания 5 В, а самый светлый *(управляющий провод)* — к контакту цифрового выхода на плате. Если у вас другой сервопривод, схему его подключения ищите в сопроводительной документации.

Управление сервоприводом

Теперь попробуем управлять сервоприводом. Следующий скетч выполняет его поворот, используя весь доступный диапазон. Подключите сервопривод к плате Arduino, как описано выше. Управляющий провод соедините с цифровым контактом 4, а затем введите и загрузите скетч в листинге 14.1.

Листинг 14.1. Скетч, демонстрирующий работу серво

```
#include <Servo.h>
Servo myservo;

void setup()
{
  myservo.attach(4);
}

void loop()
{
  myservo.write(180);
  delay(1000);
  myservo.write(90);
  delay(1000);
  myservo.write(0);
  delay(1000);
}
```

Этот скетч использует библиотеку Servo, которую нужно установить. Следуя инструкциям из главы 7, найдите в менеджере библиотек Library Manager и установите библиотеку Servo by Michael Margolis, Arduino. Он создает экземпляр класса `Servo`, как показано ниже:

```
#include <Servo.h>
Servo myservo;
```

Затем в функции `void setup()` этому экземпляру сообщается, какой контакт на плате Arduino должен использоваться для управления серво:

```
myservo.attach(4); // контакт 4 управляет серво
```

После этого серво поворачивается простой командой:

```
myservo.write(x);
```

где *x* — целое число от 0 до 180 градусов, угол, на который должен быть повернут вал сервопривода. В ходе работы скетча из листинга 14.1 сервопривод будет поворачивать вал в одну и в другую сторону, приостанавливаясь в крайних позициях (0 и 180 градусов) и в середине (90 градусов). Учтите, что позиция, соответствующая углу поворота 180 градусов, находится слева, позиция 0 градусов — справа.

Помимо перемещения объектов, сервоприводы можно использовать и для сообщения информации, как это делают аналоговые индикаторы. Например, на основе сервопривода можно сконструировать аналоговый термометр, как в проекте 37.

Проект 37: аналоговый термометр

На основе серво и уже знакомого нам температурного датчика TMP36 мы сконструируем аналоговый термометр. Он будет измерять температуру и преобразовывать ее в угол поворота от 0 до 180 градусов, соответствующий шкале температуры от 0 до +30 °C. Сервопривод будет поворачиваться на угол, соответствующий текущей температуре.

Оборудование

Нам понадобится немного:

- один температурный датчик TMP36;
- одна макетная плата;
- один небольшой серво;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Схема

Схема устройства очень проста. Ее можно увидеть на рис. 14.3.

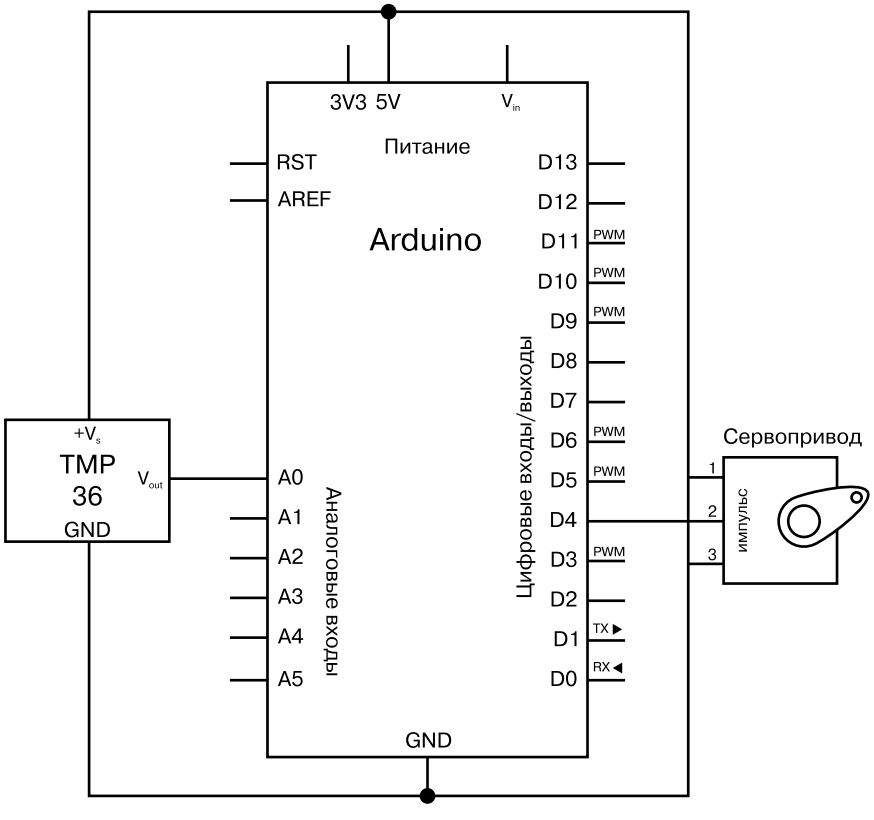


Рис. 14.3. Принципиальная схема для проекта 37

Скетч

Скетч определяет температуру с помощью приема из проекта 8 в главе 4. Затем значение температуры преобразуется в угол поворота серво.

Введите и загрузите следующий скетч:

```
// Проект 37 – аналоговый термометр

float  voltage = 0;
float   sensor = 0;
float currentC = 0;
int      angle = 0;
```

```
#include <Servo.h>
Servo myservo;

void setup()
{
  myservo.attach(4);
}

❶ int calculateservo(float temperature)
{
  float resulta;
  int resultb;
  resulta = -6 * temperature;
  resulta = resulta + 180;
  resultb = int(resulta);
  return resultb;
}

void loop()
{
  // Прочитать текущую температуру
  sensor = analogRead(0);
  voltage = (sensor*5000)/1024;
  voltage = voltage-500;
  currentC = voltage/10;

  // Преобразовать температуру в угол поворота
❶ angle = calculateservo(currentC);

  // Выполнить поворот серво
  if (angle >= 0 && angle <= 180)
  {
    myservo.write(angle); // повернуть на угол angle
    delay(1000);
  }
}
```

Большая часть скетча должна быть понятна, единственное новое здесь — это функция `calculateservo()` ❶. Она преобразует температуру в угол поворота сервопривода по следующей формуле:

$$\text{угол} = (-6 \times \text{температура}) + 180.$$

Возможно, вы захотите добавить *шкалу* с диапазоном температур и маленькую стрелку для наглядности. Пример такой шкалы показан на рис. 14.4. Изображение для печати можно получить на веб-сайте книги по адресу https://nostarch.com/arduino-workshop-2nd-edition/.

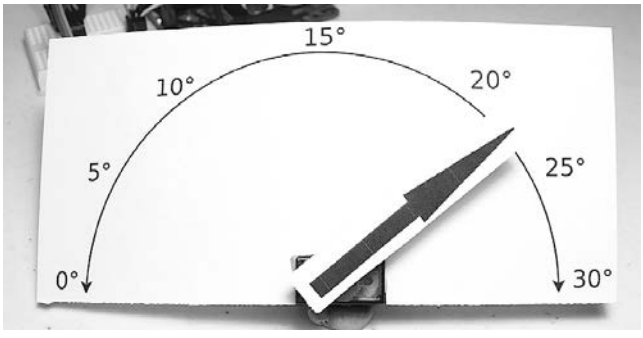


Рис. 14.4. Шкала для определения температуры, отображаемой термометром

Электродвигатели

Следующим нашим шагом будет управление маленькими электродвигателями. У маленьких экземпляров широкая область применения: от небольших вентиляторов до игрушечных автомобилей и моделей железной дороги.

Выбор электродвигателя

Как и в случае с сервоприводами, при выборе электродвигателя важно учитывать несколько параметров:

- **рабочее напряжение** — может меняться от 3 до более чем 12 В;
- **ток без нагрузки** — ток, потребляемый электродвигателем при рабочем напряжении, когда вал вращается свободно, в отсутствие любых связанных с ним механизмов;
- **пусковой ток** — нужен, чтобы провернуть вал двигателя при его запуске;
- **скорость вращения при рабочем напряжении** — скорость вращения вала в оборотах в минуту (об/мин).

В нашем примере будет использоваться маленький и недорогой электродвигатель со скоростью вращения 8540 об/мин при рабочем напряжении 3 В (рис. 14.5).

Управлять электродвигателем можно будет с помощью транзистора, как описывалось в главе 3. Так как электродвигатель потребляет ток до 0,7 А (больше, чем может пропустить транзистор BC548), для нашего проекта будет использоваться составной транзистор, который называют транзистором Дарлингтона.

Рис. 14.5. Маленький электродвигатель

Транзистор Дарлингтона TIP120

Транзистор Дарлингтона — это всего лишь два транзистора, соединенных вместе. Он может пропускать большой ток с высоким напряжением. Транзистор Дарлингтона TIP120 способен пропускать ток до 5 А с напряжением 60 В. Этого более чем достаточно для управления нашим маленьким электродвигателем. Для обозначения транзистора TIP120 на принципиальных схемах используется значок, напоминающий обозначение транзистора BC548 (рис. 14.6), но TIP120 больше, чем BC548.

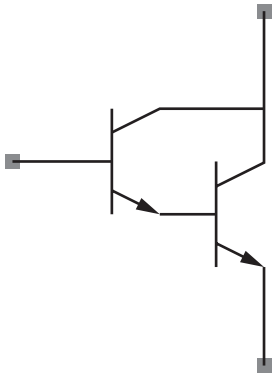


Рис. 14.6. Обозначение транзистора TIP120

Транзистор TIP120 выпускается в корпусе TO-220, как показано на рис. 14.7.

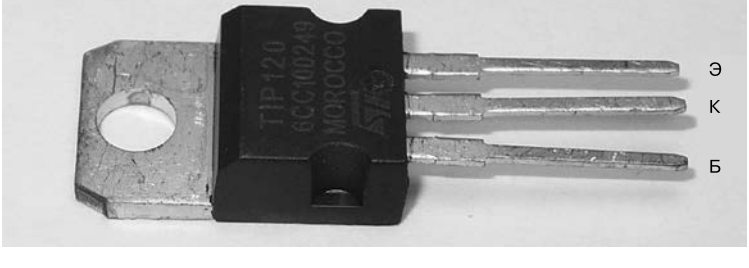


Рис. 14.7. Транзистор TIP120

Если смотреть на TIP120 со стороны маркировки, его выводы будут иметь следующее назначение (слева направо): база (Б), коллектор (К) и эмиттер (Э).

Металлический язычок для крепления транзистора на радиаторе соединен с коллектором.

Проект 38: управление электродвигателем

В этом проекте мы научимся регулировать скорость вращения электродвигателя.

Оборудование

Ниже перечислено оборудование для этого проекта:

- один маленький электродвигатель с рабочим напряжением 3 В;
- один резистор номиналом 1 кОм (R1);
- одна макетная плата;
- один диод 1N4004;
- один транзистор Дарлингтона TIP120;
- отдельный источник питания с напряжением 3 В;
- несколько отрезков провода разной длины;
- плата Arduino и кабель USB.

Для работы с электродвигателями нужен отдельный источник питания, потому что плата Arduino не способна отдавать большие токи. Если электродвигатель остановится, для повторного запуска он может потребовать *пускового тока* силой более 1 А. Это больше, чем может отдать Arduino. Если попытаться получить с платы такой ток, она может выйти из строя.

Простейшее решение — использовать батарейный контейнер. Для питания электродвигателя с рабочим напряжением 3 В достаточно контейнера на две батарейки типа AA (рис. 14.8).

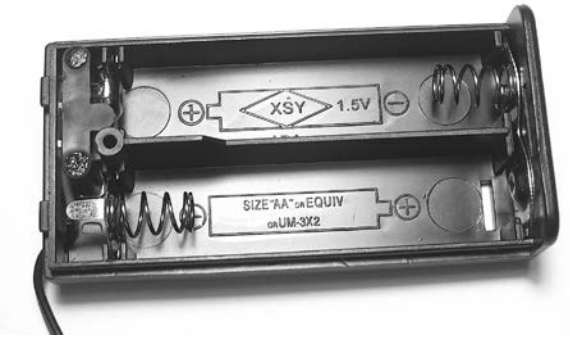


Рис. 14.8. Батарейный контейнер на две батарейки типа AA

Схема

Соберите схему, изображенную на рис. 14.9.

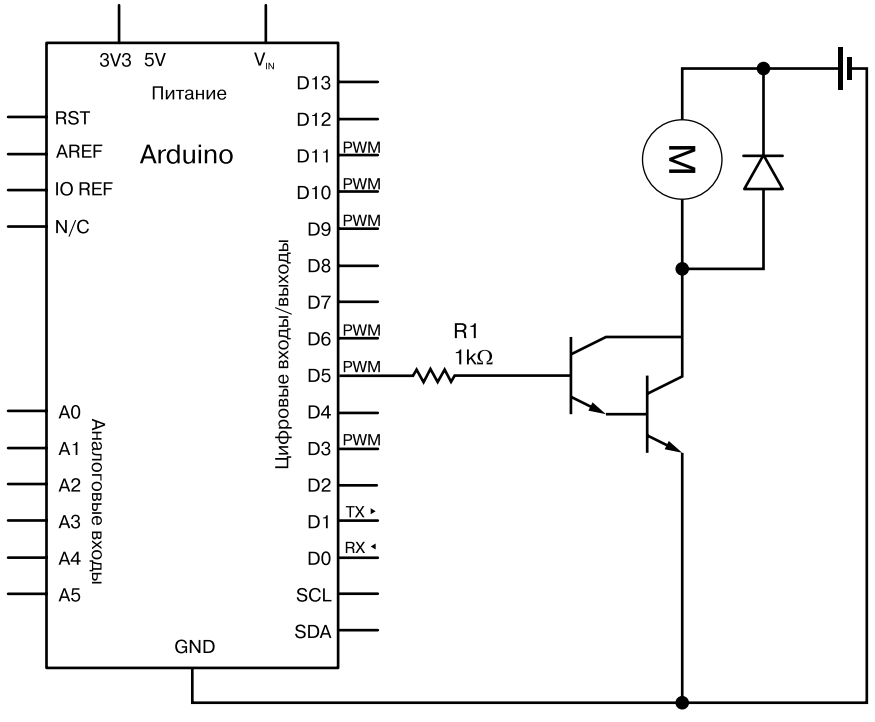


Рис. 14.9. Принципиальная схема для проекта 38

Скетч

В этом проекте мы будем регулировать скорость вращения электродвигателя от нуля (полная остановка) до максимального числа оборотов в минуту и затем обратно до полной остановки. Введите и загрузите следующий скетч:

```
// Проект 38 – управление электродвигателем

void setup()
{
  pinMode(5, OUTPUT);
}

void loop()
{
```

```
❶   for (int a=0; a<256; a++)
    {
      analogWrite(5, a);
❷    delay(100);
    }
❸   delay(5000);
❹   for (int a=255; a>=0; a--)
    {
      analogWrite(5,a);
      delay(100);
    }
    delay(5000);
  }
```

Управление скоростью вращения двигателя осуществляется с применением широтно-импульсной модуляции (как описывалось в проекте 3). Напомню, что этот метод поддерживается только цифровыми контактами 3, 5, 6, 9, 10 и 11. При использовании этого метода ток будет подаваться на электродвигатель короткими импульсами. Чем длиннее импульсы, тем выше скорость вращения (в единицу времени электродвигатель дольше будет находиться включенным). В начале первого цикла `for` ❶ электродвигатель выключается, а потом подаваемое на него напряжение начинает постепенно увеличиваться. Ускорением управляет задержка в ❷. В точке ❸ скорость вращения электродвигателя достигает максимума и удерживается так в течение пяти секунд. В следующем цикле `for` ❹ выполняется обратный процесс и электродвигатель останавливается.

ПРИМЕЧАНИЕ

Перед началом вращения электродвигателя вы можете услышать исходящее от него гудение, похожее на звук трамвая или электровоза, когда он трогается с места. Это нормально, не волнуйтесь.

Диод на этой схеме используется с той же целью, что и на схеме подключения управляющего реле (см. рис. 3.19) — для защиты схемы. Когда с электродвигателя снимается напряжение, возникает кратковременный всплеск паразитного тока высокого напряжения на его обмотке, который должен быть куда-то направлен. Диод пропускает паразитный ток по кругу через обмотку электродвигателя, пока он не будет рассеян в виде небольшого количества тепла.

Шаговые моторы

Такие моторы отличаются от обычных тем, что делят полный оборот двигателя на фиксированное количество шагов. Для этого в них используются две независимо управляемые обмотки. В отличие от обычного двигателя, где напряжение управляет скоростью вращения, в шаговом моторе можно включать и выключать

обмотки в определенном порядке, чтобы повернуть вал в любом направлении заданное количество раз. Это делает шаговые двигатели идеальными для механизмов, требующих точного позиционирования вала мотора. Они встречаются в самых разных устройствах, от компьютерных принтеров до современных промышленных станков.

Мы рассмотрим работу шагового мотора на примере модели 28BYJ-48 (рис. 14.10). Моторы этого типа позволяют поворачивать вал в одно из 4096 положений. Иначе говоря, один полный оборот делится на 4096 шагов.

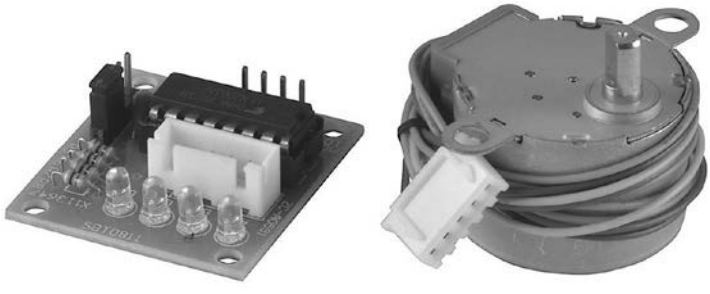


Рис. 14.10. Шаговый электродвигатель и управляющая плата

Плата рядом с двигателем служит интерфейсом между Arduino и шаговым двигателем, упрощая и ускоряя подключение. Обычно она поставляется вместе с шаговым мотором. На рис. 14.11 плата показана крупным планом.

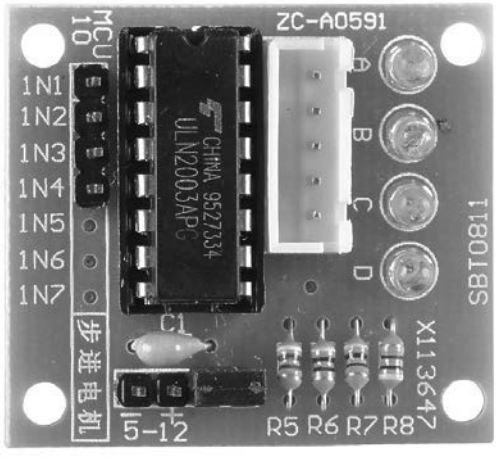


Рис. 14.11. Плата контроллера шагового мотора

Теперь подключите шаговый мотор, руководствуясь данными в табл. 14.1.

Таблица 14.1. Подключение контроллера шагового мотора к Arduino

Контакт на плате контроллера	Контакт на плате Arduino
IN1	D8
IN2	D9
IN3	D10
IN4	D11
5–12 V+	5 V
5–12 V–	GND

Шаговый мотор можно ненадолго запускать, запитав его от Arduino, если к ней больше ничего не подключено. Но лучше так не делать. Вместо этого возьмите внешний источник питания на 5 В. На плате контроллера нет разъема для подключения питания. Вы можете использовать внешний разъем с клеммами (рис. 14.12). Он позволит выполнить соединение без пайки.

Рис. 14.12. Внешний разъем с клеммами для подключения электропитания

Клеммы + и – этого разъема можно соединить проводами с контактами на плате контроллера шагового мотора. Чтобы упростить управление шаговым мотором в скетчах, дальше мы используем библиотеку для Arduino под названием CheapStepper.

Она доступна по адресу https://github.com/tyhenry/CheapStepper/archive/master.zip. Установите ее по инструкции из главы 7.

После установки библиотеки и подключения шагового мотора, как описывалось выше, введите и загрузите скетч из листинга 14.2.

Листинг 14.2. Тестирование шагового мотора

```
❶ #include <CheapStepper.h>

❷ CheapStepper stepper (8, 9, 10, 11);
❸ boolean clockwise = true;
  boolean cclockwise = false;

❹ void setup()
  {
    stepper.setRpm(20);
    Serial.begin(9600);
  }

  void loop()
  {
    Serial.println("stepper.moveTo (Clockwise, 0)");
❺   stepper.moveTo (clockwise, 0);
    delay(1000);

    Serial.println("stepper.moveTo (Clockwise, 1024)");
❺   stepper.moveTo (clockwise, 1024);
    delay(1000);

    Serial.println("stepper.moveTo (Clockwise, 2048)");
    stepper.moveTo (clockwise, 2048);
    delay(1000);

    Serial.println("stepper.moveTo (Clockwise, 3072)");
    stepper.moveTo (clockwise, 3072);
    delay(1000);

    Serial.println("stepper.moveTo (CClockwise, 512)");
❺   stepper.moveTo (cclockwise, 512);
    delay(1000);

    Serial.println("stepper.moveTo (CClockwise, 1536)");
❺   stepper.moveTo (cclockwise, 1536);
    delay(1000);

    Serial.println("stepper.moveTo (CClockwise, 2560)");
    stepper.moveTo (cclockwise, 2560);
    delay(1000);
```

```
    Serial.println("stepper.moveTo (CClockwise, 3584)");
    stepper.moveTo (cclockwise, 3584);
    delay(1000);
}
```

Управлять шаговым мотором несложно. Сначала подключите библиотеку ❶ и создайте объект ❷, представляющий мотор в программе (если вы решите использовать для управления другие цифровые выходы, укажите их здесь). Функция управления принимает в первом параметре значения `true` и `false`. Они обозначают поворот по часовой стрелке и против, поэтому в скетче определяются логические переменные ❸ для дальнейшего использования, чтобы было понятнее. Наконец, вал мотора можно повернуть в одно из 4096 положений вызовом функции:

```
Stepper.moveTo(direction, location);
```

где в параметре *`direction`* передается значение `clockwise` (`true` — по часовой стрелке) или `cclockwise` (`false` — против часовой стрелки) и в параметре *`location`* — значение от 0 до 4095. Эта функция используется в строках ❺ на протяжении всего скетча.

В `void setup()` ❹ задается скорость вращения мотора, равная 20 об/мин:

```
stepper.setRpm(20);
```

Это рекомендуемая скорость для такого шагового мотора. Для других скорость может отличаться, поэтому уточните эту деталь у поставщика.

Через несколько мгновений после загрузки скетча ваш шаговый мотор начнет поворачиваться в разных направлениях, а в мониторе порта одновременно будут появляться соответствующие команды (рис. 14.13).

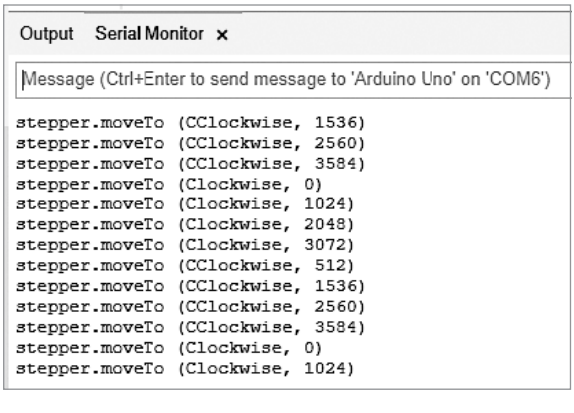


Рис. 14.13. Команды, посылаемые шаговому мотору

Проект 39: робот с электродвигателями и управление им

Умение управлять скоростью вращения одного электродвигателя постоянного тока может пригодиться при реализации проектов. Но я предлагаю заняться кое-чем поинтереснее — управлением скоростью *и* направлением вращения сразу четырех электродвигателей. Наша цель — создать робота с четырьмя ведущими колесами, работать над которыми мы будем следующие несколько глав. Здесь я только опишу конструкцию и основы управления роботом.

Итак, у робота есть четыре электродвигателя. Каждый из них управляет одним колесом. Он может двигаться с разной скоростью и разворачиваться на месте. Вы научитесь управлять скоростью и направлением движения и узнаете, как добавить в его конструкцию новые детали, предотвращающие столкновения и позволяющие дистанционно управлять роботом. Все проекты из этой книги помогут вам получить знания и навыки для создания собственных версий робота и воплощения новых идей.

Оборудование

Для создания робота нужны следующие компоненты:

- одно шасси робота с четырьмя электродвигателями и колесами;
- батарейный блок для четырех батареек типа AA;
- четыре алкалиновые батарейки типа AA;
- плата расширения на базе микросхем L293D для управления электродвигателями;
- плата Arduino и кабель USB.

Шасси

Основа любого робота — шасси с электродвигателями, трансмиссией и источником электропитания. Кроме того, у робота, управляемого платой Arduino, должно быть место для размещения платы и других дополнительных компонентов.

На рынке вы найдете множество шасси для моделирования. Но мы будем использовать недорогое шасси для конструирования роботов, в комплект которого входят четыре электродвигателя с рабочим напряжением питания около 6 В и колеса (рис. 14.14).

Порядок сборки шасси робота зависит от модели. Для этого могут понадобиться дополнительные инструменты — отвертки и плоскогубцы. Если в будущем вы хотите доработать конструкцию, а пока вам просто нужно получить движущуюся модель робота, лучше всего закрепить электронику на шасси офисным пластилином Blu Tack.

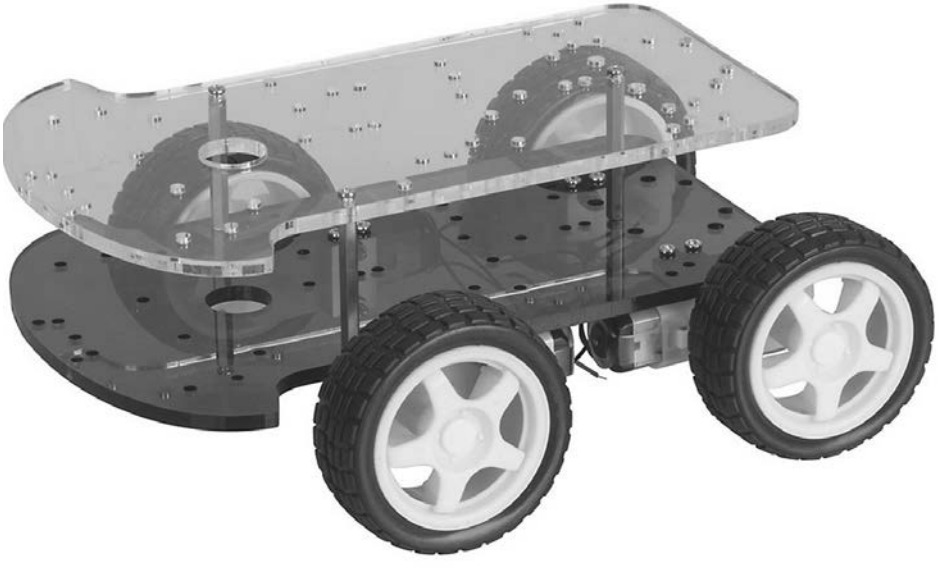


Рис. 14.14. Шасси для нашего робота

Источник питания

Поставляемые в комплекте с шасси электродвигатели питаются постоянным током напряжением около 6 В. Поэтому для питания робота мы будем использовать контейнер на четыре батарейки типа AA (рис. 14.15).

Рис. 14.15. Контейнер на четыре батарейки типа AA

У некоторых контейнеров нет провода, которым можно было бы подключить его к нашей конструкции, а оборудуются они, например, защелкивающимся разъемом для батареи 9 В (как на нашем контейнере на рис. 14.15). В этом случае вам нужен ответный разъем с проводом (рис. 14.16).

Рис. 14.16. Провод с разъемом для подключения Arduino к контейнеру с батареями

Схема

Теперь нужно собрать схему управления четырьмя электродвигателями на шасси. Можно было бы для каждого электродвигателя использовать схему с рис. 14.9, но она не позволяет выбирать направление вращения электродвигателей. К тому же ее неудобно собирать. Поэтому мы воспользуемся специальной *платой расширения для управления электродвигателями*. В ней есть все необходимое для подачи большого тока на электродвигатели. А еще она умеет принимать от платы Arduino команды управления скоростью и направлением вращения электродвигателей. В нашей модели робота будет использоваться плата на базе L293D для Arduino (рис. 14.17).

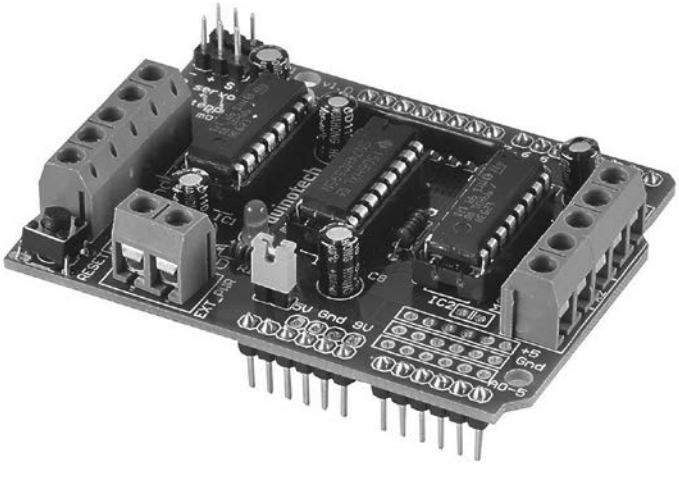


Рис. 14.17. Плата управления на базе микросхем L293D

Подключение платы управления электродвигателями

Это довольно просто: подключите провода от контейнера с батареями к блоку контактов на плате слева внизу (рис. 14.18). Черный провод (минус) должен подключаться к правому контакту, а красный (плюс) — к левому.

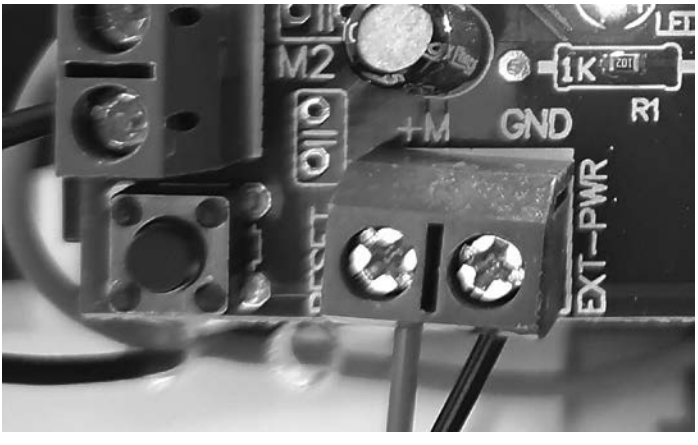


Рис. 14.18. Подключение проводов питания

Теперь подключите каждый электродвигатель к плате управления. Далее электродвигатели будут определяться номерами: два спереди будут иметь номера 1 (справа) и 2 (слева), а два сзади — 3 (слева) и 4 (справа). От каждого электродвигателя отходят красный и черный провода. Подключите их к соответствующим клеммным колодкам с левой и правой стороны платы управления, как на рис. 14.19.

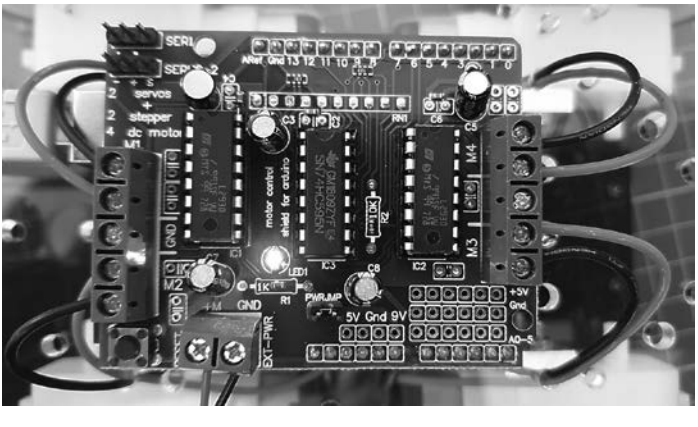


Рис. 14.19. Подключение электродвигателей

При подключении электродвигателей обратите внимание, что черные провода подключены к внешним клеммам, а красные — к внутренним. Для удобства каждая клеммная колодка подписана соответствующим номером двигателя (M1, M2, M3 и M4).

Если идущие от электродвигателей провода одного цвета, после первой попытки может выясниться, что двигатели вращаются не в том направлении. В таком случае вам может понадобиться поменять их местами в клеммах на плате.

После подключения контейнера для батарей, электродвигателей к плате управления, а платы управления к Arduino робот должен выглядеть так, как показано на рис. 14.20.

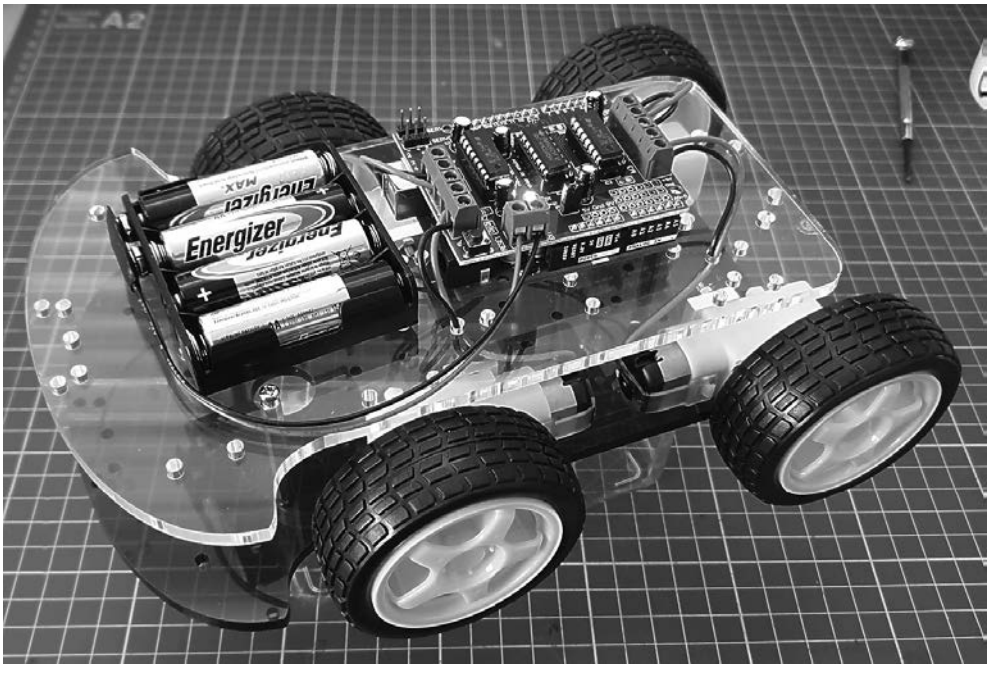


Рис. 14.20. Робот готов к испытаниям!

Скетч

Теперь заставим робота двигаться. Чтобы упростить управление, сначала загрузите и установите библиотеку поддержки платы управления электродвигателями. Как это сделать, вы можете узнать из главы 7. В менеджере библиотек Library Manager отыщите и установите библиотеку Adafruit Motor Shield от Adafruit.

Начинайте вводить название библиотеки в поле поиска, и спустя короткое время Adafruit Motor Shield library v1 появится в списке. Нажмите Install (Установить) и подождите, пока библиотека установится. После этого окно менеджера библиотек можно закрыть.

Теперь создадим несколько функций, которые потом упростят управление роботом. Так как робот приводится в движение двумя парами электродвигателей, нам нужно реализовать четыре вида движений:

- движение вперед;
- движение назад;
- поворот по часовой стрелке;
- поворот против часовой стрелки.

Четырем видам движений в скетче соответствуют четыре функции: `goForward()`, `goBackward()`, `rotateLeft()` и `rotateRight()`. Каждая принимает значение в миллисекундах, определяющее время, в течение которого будут включены электродвигатели, и скорость в виде значения ШИМ от 0 до 255. Например, чтобы заставить робота двигаться вперед на полной скорости в течение двух секунд, нужно выполнить вызов `goForward(2000, 255)`.

Введите и сохраните следующий скетч (но пока не загружайте его):

```
// Проект 39 — робот с электродвигателями и управление им
#include <AFMotor.h>

❶ AF_DCMotor motor1(1); // Настройка экземпляров, представляющих моторы
AF_DCMotor motor2(2);
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);

❷ void goForward(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}
```

```
❷ void goBackward(int speed, int duration)
  {
    motor1.setSpeed(speed);
    motor2.setSpeed(speed);
    motor3.setSpeed(speed);
    motor4.setSpeed(speed);
    motor1.run(BACKWARD);
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
    motor4.run(BACKWARD);
    delay(duration);
    motor1.run(RELEASE);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor4.run(RELEASE);
  }

❷ void rotateLeft(int speed, int duration)
  {
    motor1.setSpeed(speed);
    motor2.setSpeed(speed);
    motor3.setSpeed(speed);
    motor4.setSpeed(speed);
    motor1.run(FORWARD);
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
    motor4.run(FORWARD);
    delay(duration);
    motor1.run(RELEASE);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor4.run(RELEASE);
  }

❷ void rotateRight(int speed, int duration)
  {
    motor1.setSpeed(speed);
    motor2.setSpeed(speed);
    motor3.setSpeed(speed);
    motor4.setSpeed(speed);
    motor1.run(BACKWARD);
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(BACKWARD);
    delay(duration);
    motor1.run(RELEASE);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor4.run(RELEASE);
  }
```

```
void setup()
{
  delay(5000);
}

void loop()
{
  goForward(127,5000);
  delay(1000);
  rotateLeft(127,2000);
  delay(1000);
  goBackward(127,5000);
  delay(1000);
  rotateRight(127,5000);
  delay(5000);
}
```

Четыре написанные нами функции упрощают управление роботом. Все они используют функции из библиотеки поддержки платы управления электродвигателями. Но, прежде чем использовать их, нужно создать экземпляр класса, представляющий каждый электродвигатель, как показано в ❶.

Направление вращения каждого электродвигателя задается вызовом:

```
Motor.run(direction)
```

В параметре *direction* можно передать значение `FORWARD`, `REVERSE` или `RELEASE`, чтобы заставить электродвигатель вращаться в прямом или обратном направлении либо остановиться.

Скорость вращения задается вызовом:

```
Motor.setSpeed(speed)
```

Параметр *speed* может принимать значения от 0 до 255. Это диапазон ШИМ для управления скоростью вращения электродвигателя.

В каждой из наших функций ❷ используется комбинация управляющих воздействий, определяющих скорость и направление вращения всех четырех двигателей. Каждая функция принимает два параметра: `speed` (скорость — значение ШИМ) и `duration` (продолжительность — время, в течение которого все электродвигатели будут оставаться включенными).

ВНИМАНИЕ

Когда вы будете готовы загрузить скетч, поднимите робота над столом, чтобы его колеса свободно вращались в воздухе. Если этого не сделать, после загрузки скетча робот двинется вперед и может упасть на пол!

Загрузите скетч, отключите кабель USB и соедините плату Arduino с батарейным контейнером. Теперь поставьте робота на ковер или другую поверхность без посторонних предметов и отпустите. Поэкспериментируйте с функциями движения в скетче. Это поможет вам лучше понять суть задержек и их влияние на величину пройденного расстояния.

Подключение дополнительного оборудования к роботу

В некоторых платах расширения для управления моторами могут отсутствовать разъемы, позволяющие подключить одну плату поверх другой. В этом случае вы не сможете легко подключить датчики и т. д. Для этого можно использовать терминальный адаптер для Arduino (рис. 14.21).

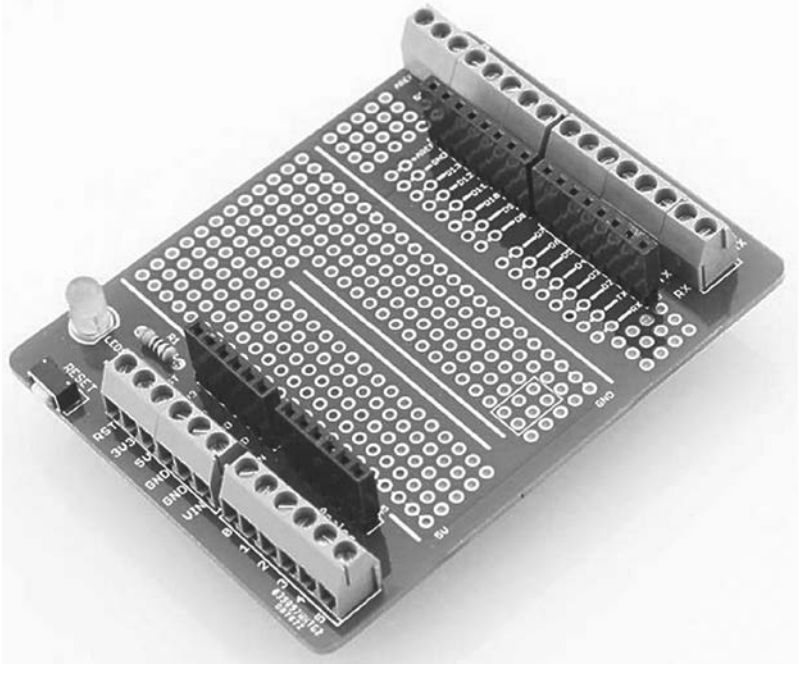


Рис. 14.21. Терминальный адаптер для Arduino

С помощью терминального адаптера можно легко подключать оборудование или датчики к входам и выходам Arduino без всякой пайки. Его можно применять и при сборке своей схемы для постоянного использования в других проектах позднее.

Определение столкновений

Мы научили робота двигаться. Теперь добавим датчики, которые сообщат ему о препятствии, когда тот упрется в него, или измерят расстояние между роботом и преградой, чтобы тот мог повернуть и избежать столкновения. Мы будем использовать три способа предотвращения столкновений: микровыключатели, инфракрасные и ультразвуковые датчики.

Проект 40: определение столкновений с помощью микровыключателя

Микровыключатель действует как обычная кнопка без фиксации из главы 4. Разница в том, что микровыключатель больше и имеет длинную металлическую пластину, играющую роль рычага (рис. 14.22).

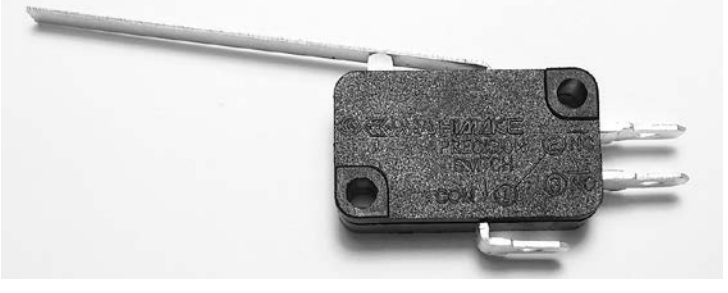


Рис. 14.22. Микровыключатель

Обычно при использовании микровыключателя один провод подключается к нижнему выводу, а другой — к выводу с меткой NO (normally open — «нормально разомкнутый»), чтобы ток протекал только при нажатом рычаге. Мы смонтируем микровыключатель в передней части робота. Когда тот достигнет какого-то препятствия, рычаг замкнет микровыключатель, ток потечет через его выводы и заставит робота изменить направление движения.

Схема

Подсоедините микровыключатель как обычную кнопку (рис. 14.23).

Скетч

Мы подключили микровыключатель к цифровому контакту 2, поддерживающему прерывания. Может показаться, что мы должны использовать функцию обработки прерываний, чтобы заставить робота какое-то время двигаться в обратном

направлении. Но это невозможно, потому что функция `delay()` не работает внутри обработчиков прерываний. Здесь нужно искать другое решение.

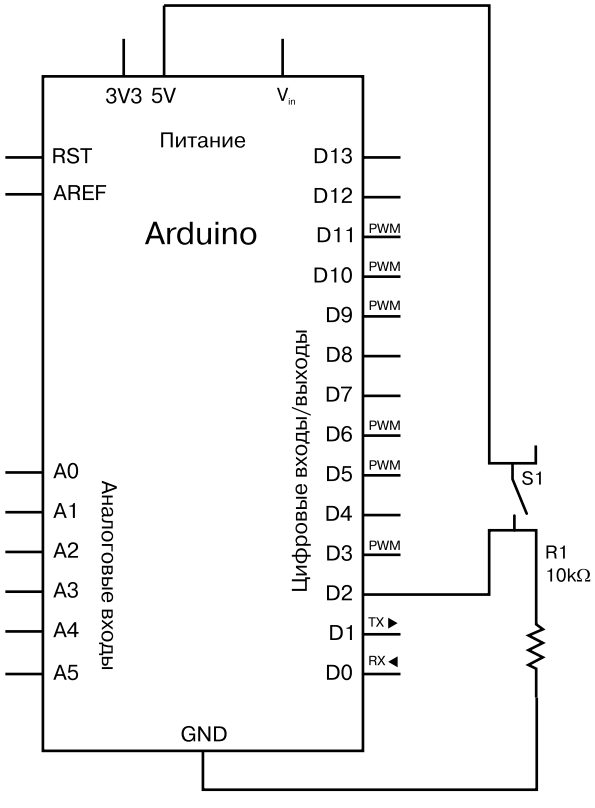


Рис. 14.23. Схема для проекта 40

Оно состоит в следующем: функция `goForward()` будет включать электродвигатели только при соблюдении двух условий, задаваемых логическими переменными `crash` и `move`. Если `crash` имеет значение `true`, электродвигатели будут на две секунды включаться в обратном направлении с низкой скоростью, чтобы «выйти» из соприкосновения с препятствием.

Мы не можем вызывать функцию `delay()` в обработчике прерывания, поэтому время работы электродвигателей будет измеряться иначе. Сначала будет вызываться `millis()`, чтобы засечь момент включения двигателей, а затем в цикле каждое новое ее значение будет сравниваться с начальным. Когда разность сравняется или превысит требуемую продолжительность, функция присвоит переменной `move` значение `false` и остановит электродвигатели.

Введите и загрузите следующий скетч:

```
// Проект 40 – определение столкновений с помощью микровыключателя

#include <AFMotor.h>

AF_DCMotor motor1(1); // Настройка экземпляров, представляющих моторы
AF_DCMotor motor2(2);
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);

boolean crash=false;

void goBackward(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

❶ void backOut()
{
  crash = true;
}

void goForward(int duration, int speed)
{
  long a, b;
  boolean move = true;
❷ a = millis();
  do
  {
    if (crash == false)
    {
      motor1.setSpeed(speed);
      motor2.setSpeed(speed);
      motor3.setSpeed(speed);
      motor4.setSpeed(speed);
      motor1.run(FORWARD);
      motor2.run(FORWARD);
      motor3.run(FORWARD);
```

```
      motor4.run(FORWARD);
    }
    if (crash == true)
    {
❸     goBackward(200, 2000);
      crash = false;
    }
❹   b = millis() - a;
    if (b >= duration)
    {
      move = false;
    }
  }
  while (move != false);

  // Выключить моторы
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void setup()
{
  attachInterrupt(0, backOut, RISING);
  delay(5000);
}

void loop()
{
  goForward(5000, 127);
  delay(2000);
}
```

В этом скетче реализован усовершенствованный алгоритм движения вперед. Управление движением происходит с применением двух переменных. Первая — логическая переменная `crash`. Если робот наткнулся на какое-то препятствие и включил микровыключатель, возникает прерывание, для обработки которого вызывается функция `backOut()` ❶. Она присваивает `crash` значение `true`. Вторая управляющая движением переменная — логическая переменная `move`. Функция `goForward()` вызывает `millis()` ❷, чтобы засечь время начала движения. По нему потом будет определяться момент истечения времени движения робота (устанавливается параметром `duration`).

В строке ❹ функция вычисляет время от начала движения. Если оно меньше заданной продолжительности, в переменной `move` сохраняется значение `true`. То есть роботу можно продолжить движение вперед, только если он не столкнулся

с препятствием и время движения не истекло. При обнаружении столкновения вызывается функция `goBackward()` ❸ и робот начинает медленно двигаться в обратном направлении в течение двух секунд, а потом едет как обычно.

ПРИМЕЧАНИЕ

Чтобы расширить или изменить этот пример, попробуйте добавить в него другие функции движения из проекта 39.

Инфракрасный датчик расстояния

Наш следующий метод основан на применении инфракрасного (ИК) датчика расстояния. Он фиксирует ИК-излучение, отраженное от поверхности перед ним, и возвращает напряжение, пропорциональное расстоянию между датчиком и поверхностью. Инфракрасные датчики расстояния удобно использовать для определения препятствий. Они недорогие, но не годятся для *точного* измерения расстояний. В следующем проекте мы используем аналоговый датчик Sharp GP2Y0A21YK0F (рис. 14.24).

Рис. 14.24. ИК-датчик Sharp

Подключение

Чтобы подключить датчик к плате Arduino, соедините красный и черный провода от датчика с контактами 5 V и GND, а белый — с аналоговым входом. Для измерения возвращаемого датчиком напряжения мы будем использовать функцию `analogRead()`. График на рис. 14.25 показывает зависимость между расстоянием и выходным напряжением.

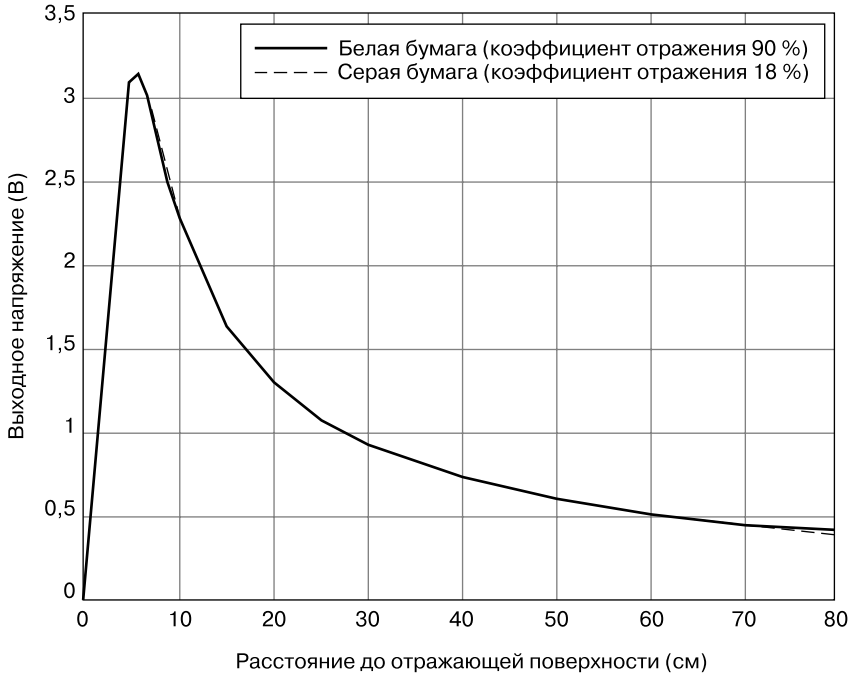


Рис. 14.25. График зависимости выходного напряжения на ИК-датчике от расстояния

Тестирование ИК-датчика расстояния

Зависимость между расстоянием и выходным напряжением трудно представить уравнением, поэтому мы разобьем график на пятисантиметровые отрезки. Рассмотрим, как это делается, на простом примере. Подключите белый провод инфракрасного датчика к аналоговому контакту 0, красный провод — к контакту 5 V и черный — к GND. Затем введите и загрузите скетч из листинга 14.3.

Листинг 14.3. Скетч, демонстрирующий использование ИК-датчика

```
float sensor = 0;
int cm = 0;

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

void loop()
{
```

```
❶   sensor = analogRead(0);
❷   if (sensor<=90)
    {
      Serial.println("Infinite distance!");
    } else if (sensor<100) // 80 см
    {
      cm = 80;
    } else if (sensor<110) // 70 см
    {
      cm = 70;
    } else if (sensor<118) // 60 см
    {
      cm = 60;
    } else if (sensor<147) // 50 см
    {
      cm = 50;
    } else if (sensor<188) // 40 см
    {
      cm = 40;
    } else if (sensor<230) // 30 см
    {
      cm = 30;
    } else if (sensor<302) // 25 см
    {
      cm = 25;
    } else if (sensor<360) // 20 см
    {
      cm = 20;
    } else if (sensor<505) // 15 см
    {
      cm = 15;
    } else if (sensor<510) // 10 см
    {
      cm = 10;
    } else if (sensor>=510) // слишком близко!
    {
      Serial.println("Too close!");
    }
    Serial.print("Distance: ");
    Serial.print(cm);
    Serial.println(" cm");
    delay(250);
  }
```

Скетч читает напряжение с ИК-датчика ❶, а потом с помощью последовательности инструкций `if` ❷ определяет примерное расстояние. Мы определяем его по напряжению, возвращаемому датчиком, используя два параметра. Во-первых, это отношение напряжения к расстоянию, как на рис. 14.25. Далее, используя знания (из проекта 6 в главе 4), что функция `analogRead()` возвращает значение от 0 до 1023 относительно

напряжения от 0 до примерно 5 В, мы можем рассчитать приблизительное расстояние, возвращаемое датчиком.

После загрузки скетча откройте окно монитора порта и поэкспериментируйте, приближая руку или лист бумаги к датчику и отдаляя их от него. В это время в окне монитора порта должно выводиться приблизительное расстояние, как показано на рис. 14.26.

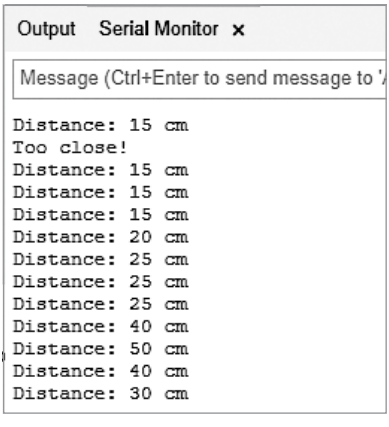


Рис. 14.26. Результаты работы скетча из листинга 14.3

Проект 41: определение столкновений с помощью ИК-датчика расстояния

Теперь заменим микровыключатель инфракрасным датчиком расстояния. Далее мы будем использовать немного измененную версию проекта 40. Вместо обработки прерывания добавим в скетч функцию `checkDistance()`, которая будет присваивать переменной `crash` значение `true`, если расстояние до препятствия меньше 20 см. Эта функция вызывается в `goForward()` в цикле `do...while`.

Скетч

Смонтируйте ИК-датчик на шасси робота и подключите его. Далее введите и загрузите следующий скетч:

```
// Проект 41 – определение столкновений с помощью ИК-датчика расстояния

AF_DCMotor motor1(1); // Настройка экземпляров, представляющих моторы
AF_DCMotor motor2(2);
```

```
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);

boolean crash = false;

void goBackward(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void checkDistance()
{
❶ if (analogRead(0) > 460)
  {
    crash = true;
  }
}

void goForward(int duration, int speed)
{
  long a, b;
  boolean move = true;
  a = millis();
  do
  {
    checkDistance();
    if (crash == false)
    {
      motor1.setSpeed(speed);
      motor2.setSpeed(speed);
      motor3.setSpeed(speed);
      motor4.setSpeed(speed);
      motor1.run(FORWARD);
      motor2.run(FORWARD);
      motor3.run(FORWARD);
      motor4.run(FORWARD);
    }
    if (crash == true)
    {
```

```
      goBackward(200, 2000);
      crash = false;
    }
    b = millis() - a;
    if (b >= duration)
    {
      move = false;
    }
  }
  while (move != false);

  // Выключить моторы
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void setup()
{
  delay(5000);
}

void loop()
{
  goForward(5000, 255);
  delay(2000);
}
```

В этом скетче используется тот же алгоритм, что и в проекте 40. Разница только в том, что эта версия постоянно проверяет расстояние ❶ и присваивает переменной `crash` значение `true`, если расстояние между ИК-датчиком и препятствием меньше примерно 20 см.

Доработка скетча: добавление датчиков

Запустив робота, вы наверняка обратили внимание на преимущество использования бесконтактного датчика столкновений. Теперь вы без проблем сможете установить дополнительные датчики на шасси робота, например спереди и сзади или по углам корпуса. Еще вы сможете добавить код, проверяющий датчики по очереди и принимающий решения на основе информации о расстоянии до препятствия.

Ультразвуковой датчик расстояния

Рассмотрим последний способ предотвращения столкновений — применение *ультразвукового датчика расстояния*. Он посылает ультразвуковые импульсы (неслышимые человеческому уху) и измеряет время, за которое импульс вернется

назад. В этом проекте будет использован ультразвуковой датчик расстояния HC-SR04 (рис. 14.27). Он относительно недорогой и имеет точность измерений до 2 см.

Рис. 14.27. Ультразвуковой датчик расстояния HC-SR04

Указанная точность измерений обеспечивается ультразвуковым датчиком на расстояниях от 2 до 450 см. Но из-за того, что звуковая волна должна отражаться от препятствий так, чтобы вернуться к датчику, угол между направлением движения и направлением датчика не должен быть меньше 15 градусов.

Подключение ультразвукового датчика

Присоедините выводы Vcc (5 В) и GND датчика к одноименным контактам на плате управления электродвигателями, вывод Trig — к цифровому контакту D2 на плате Arduino, а вывод Echo — к цифровому контакту D13. Контакты D2 и D13 используются потому, что они не задействованы на плате управления электродвигателями. Но если вы решите просто поэкспериментировать с ультразвуковым датчиком без подключения любых других плат расширения, то можете присоединить все провода сразу к плате Arduino.

Для упрощения работы с датчиком загрузите библиотеку по адресу https://github.com/Martinsos/arduino-lib-hc-sr04/archive/master.zip и установите ее, как описано в главе 7. После установки библиотеки можете опробовать тестовый скетч из листинга 14.4 и посмотреть, как работает датчик.

Листинг 14.4. Скетч, демонстрирующий работу ультразвукового датчика

```
#include <HCSR04.h>

❶ UltraSonicDistanceSensor HCSR04(2, 13); // trig – D2, echo – D13
❷ float distance;

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

void loop ()
{
❸   distance = HCSR04.measureDistanceCm();
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(500);
}
```

Библиотека упрощает чтение расстояния из датчика. В ❶ создается экземпляр класса и указывается, к каким цифровым контактам подключен датчик. В ❷ объявляется вещественная переменная для хранения расстояния, возвращаемого библиотекой поддержки датчика. В ❸ выполняется чтение расстояния из датчика для вывода в монитор порта.

Тестирование ультразвукового датчика расстояния

После загрузки скетча откройте окно монитора порта и поэкспериментируйте, приближая какой-нибудь объект к датчику или отдаляя этот объект от него. В окне монитора порта должна при этом выводиться оценка расстояния в сантиметрах, как на рис. 14.28.

Рис. 14.28. Результаты работы скетча из листинга 14.4

Проект 42: определение столкновений с помощью ультразвукового датчика расстояния

Теперь вы знаете, как действует датчик. Попробуем использовать его на нашем роботе.

Скетч

Следующий скетч измеряет расстояние до препятствия, и если оно меньше 5 см, то включает задний ход. Введите и загрузите скетч, чтобы убедиться в правильности его работы:

```
// Проект 42 – определение столкновений
// с помощью ультразвукового датчика расстояния
#include <AFMotor.h>
#include <HCSR04.h>

// Настройка экземпляров, представляющих моторы
AF_DCMotor motor1(1);
AF_DCMotor motor2(2);
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);

// Настройка ультразвукового датчика
UltraSonicDistanceSensor HCSR04(2, 13); // trig – D2, echo – D13

boolean crash=false;

void checkDistance()
{
  float distance;
  distance = HCSR04.measureDistanceCm();
❶ if (distance < 5) // Расстояние до препятствия меньше 5 см
  {
    crash = true;
  }
}

void goBackward(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
  delay(duration);
  motor1.run(RELEASE);
```

```
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void goForward(int duration, int speed)
{
  long a, b;
  boolean move = true;
  a = millis();
  do
  {
    checkDistance();
    if (crash == false)
    {
      motor1.setSpeed(speed);
      motor2.setSpeed(speed);
      motor3.setSpeed(speed);
      motor4.setSpeed(speed);
      motor1.run(FORWARD);
      motor2.run(FORWARD);
      motor3.run(FORWARD);
      motor4.run(FORWARD);
    }
    if (crash == true)
    {
      goBackward(200, 2000);
      crash = false;
    }
    b = millis() - a;
    if (b >= duration)
    {
      move = false;
    }
  }
  while (move != false);

  // Выключить электродвигатели
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void setup()
{
  delay(5000);
}

void loop()
{
  goForward(1000, 255);
}
```

Как видите, этот скетч очень похож на предыдущий. Он так же постоянно измеряет расстояние до препятствия ❶ и присваивает переменной `crash` значение `true`, если расстояние между ультразвуковым датчиком и препятствием оказывается меньше 5 см. Наблюдать за тем, как робот будто по волшебству объезжает препятствия или состязается в сообразительности с питомцами, может быть очень увлекательно.

Что дальше?

В этой главе вы узнали, как добавить в проекты на основе Arduino возможность перемещаться в окружающем мире. С помощью моторов или простых электродвигателей с платой управления вы сможете создавать устройства, способные самостоятельно перемещаться и даже объезжать препятствия. Для демонстрации диапазона возможностей мы использовали три типа датчиков с разной стоимостью и точностью измерений. Полученные знания позволят вам принимать обоснованные решения, исходя из потребностей и бюджета проекта.

Надеюсь, теперь у вас достаточно опыта, чтобы получать удовольствие от возможности проектировать и конструировать разные устройства. Но это еще не конец. В следующей главе мы выйдем на улицу и опробуем навигацию по спутникам.

15

Arduino и GPS

В этой главе вы:

- узнаете, как подключить плату расширения GPS;
- создадите простой дисплей для отображения координат GPS;
- научитесь определять местоположение на карте по координатам GPS;
- создадите часы точного времени;
- научитесь записывать последовательность координат объекта в процессе перемещения.

Сейчас вы узнаете, как с помощью недорогой платы расширения GPS определять местоположение, как создать часы, показывающие точное время, и как сохранить траекторию движения в карте памяти microSD, которую можно наложить на карту.

Что такое GPS

Глобальная система определения местоположения (Global Positioning System, GPS) — это навигационная система, принимающая с помощью приемника GPS на Земле данные со спутников на околоземной орбите. Потом их можно будет использовать для определения текущих координат в любой точке планеты. Возможно, вы уже знакомы с навигаторами GPS в автомобилях и сотовых телефонах.

С применением Arduino нельзя создать навигационную систему с подробной картой. Но мы можем использовать модуль GPS и с его помощью определять географические координаты, время и примерную скорость перемещения (во время движения). В продаже обычно доступны две модели приемников GPS. Первая — недорогой и независимый с внешней антенной (рис. 15.1).

Рис. 15.1. GPS-приемник

Вторая распространенная модель — плата расширения GPS для Arduino, изображенная на рис. 15.2. Эти платы удобны тем, что все необходимые соединения уже выполнены. Кроме того, в них есть слот для карты памяти microSD, которая идеально подходит для записи данных. Это будет показано далее.

Перед покупкой убедитесь, что плата расширения GPS позволяет подключить линии приема/передачи (TX и RX) приемника GPS к цифровым контактам D2 и D3 на Arduino или имеет перемычки для их подключения вручную (как плата расширения на рис. 15.2). За подробностями обращайтесь к поставщику.

Рис. 15.2. Плата расширения GPS для Arduino

В этой главе вы можете использовать любую модель. Но я настоятельно рекомендую приобрести именно плату расширения. Тем более что поверх платы GPS можно подключить плату с жидкокристаллическим индикатором, который будет играть роль дисплея.

ИСПОЛЬЗОВАНИЕ ПРОГРАММНОГО ПОСЛЕДОВАТЕЛЬНОГО ПОРТА

Для работы с модулями GPS, передающими данные через последовательный порт, нужно использовать «программный последовательный порт». Наряду с обычным последовательным портом на плате Arduino, который подключен к цифровым контактам D0 и D1, можно выделить два других цифровых вывода. Они будут играть роль еще одного последовательного порта. Дополнительный порт эмулируется программно с помощью библиотеки SoftwareSerial Arduino. Он позволяет одновременно использовать два устройства, передающих данные через последовательное соединение.

Создать программный последовательный порт просто: сначала подключите библиотеку, а затем создайте еще один порт:

```
#include <SoftwareSerial.h>
SoftwareSerial Serial2(x, y);
```

где x — цифровой контакт для передачи (TX), а y — цифровой контакт для приема (RX).

После этого программный последовательный порт можно использовать точно так же, как аппаратный. Например, инициализировать его в функции `void setup()`:

```
Serial2.begin(9600);
```

Особенности использования программного последовательного порта вы сможете увидеть в скетчах в этой главе.

Тестирование платы расширения GPS

После покупки платы GPS убедитесь, что она исправна и способна принимать GPS-сигналы. Приемник GPS должен находиться в открытом пространстве, чтобы получать сигналы со спутников. Впрочем, они могут проходить и через стекло. Тестирование лучше всего производить вне помещения или хотя бы у ничем не загороженного окна. Чтобы проверить прием сигналов, введите и загрузите простой скетч, отображающий принимаемые данные в необработанном виде.

Если вы используете плату расширения GPS, убедитесь, что ее контакт TX подключается к контакту D2 на плате Arduino, а контакт RX — к контакту D3. Если вы используете внешний GPS, как на рис. 15.1, подключите его контакты Vcc и GND к контактам 5 V и GND на плате Arduino, контакт TX — к контакту D2 и RX — к D3.

Введите и загрузите скетч из листинга 15.1.

Листинг 15.1. Простой скетч для тестирования GPS

```
#include <SoftwareSerial.h>

// GPS TX к D2, RX к D3
SoftwareSerial Serial2(2, 3);

byte gpsData;

void setup()
{
  // Открыть монитор порта
  Serial.begin(9600);
  // Открыть программный порт для связи с GPS
❶ Serial2.begin(9600);
}

void loop()
{
❷ while (Serial2.available() > 0)
  {
    // Получить байт от платы GPS
    gpsData = Serial2.read();
❸   Serial.write(gpsData);
  }
}
```

Этот скетч «прослушивает» программный последовательный порт ❷. Когда модуль или плата GPS присылает байт данных, он выводит его в монитор порта ❸ (заметьте, что программный последовательный порт настраивается на работу на скорости 9600 бод ❶. Эта скорость должна совпадать со скоростью передачи данных приемником GPS).

После загрузки скетча вам придется немного подождать, пока приемник GPS поймает сигналы от одного или нескольких GPS-спутников. У плат и модулей GPS обычно есть встроенный светодиод, который начинает мигать, как только приемнику удастся получить сигнал. Когда светодиод начнет мигать, откройте окно последовательного порта в IDE и установите скорость обмена данными 9600 бод. После этого в окне отобразится непрерывный поток данных (рис. 15.3).

```
$GPRMC,002807.00,A,2734.93850,S,15305.76781,E,0.013,,030820,,,A*63
$GPVTG,,T,,M,0.013,N,0.024,K,A*27
$GPGGA,002807.00,2734.93850,S,15305.76781,E,1,08,0.95,42.4,M,38.3,M,,*7A
$GPGSA,A,3,22,03,04,26,09,16,07,27,,,,,1.81,0.95,1.55*08
$GPGSV,3,1,12,01,04,346,,03,63,355,30,04,58,177,33,06,07,244,10*70
$GPGSV,3,2,12,07,26,282,35,08,06,029,12,09,33,222,25,16,56,099,33*78
$GPGSV,3,3,12,22,40,010,28,26,26,134,28,27,10,060,32,30,00,294,*76
$GPGLL,2734.93850,S,15305.76781,E,002807.00,A,A*71
$GPRMC,002808.00,A,2734.93856,S,15305.76783,E,0.026,,030820,,,A*6E
$GPVTG,,T,,M,0.026,N,0.049,K,A*2A
$GPGGA,002808.00,2734.93856,S,15305.76783,E,1,08,0.95,42.5,M,38.3,M,,*70
$GPGSA,A,3,22,03,04,26,09,16,07,27,,,,,1.81,0.95,1.55*08
$GPGSV,3,1,12,01,04,346,,03,63,355,29,04,58,177,33,06,07,244,09*70
$GPGS
```

Рис. 15.3. Необработанные данные от приемника GPS

Приемник GPS посылает данные в плату Arduino по одному символу. Они тут же передаются в монитор порта. Но от таких необработанных данных (их еще называют *пакетами данных GPS*) мало проку. Поэтому мы задействуем новую библиотеку и с ее помощью приведем эти данные в пригодный для использования вид. Для этого загрузите и установите библиотеку TinyGPS: http://www.arduiniana.org/libraries/tinygps/, следуя процедуре, описанной в главе 7.

Проект 43: простой приемник GPS

Начнем с создания простого приемника GPS. Но так как устройства GPS обычно используются в открытом пространстве и чтобы упростить работу, добавим модуль ЖКИ для отображения данных (рис. 15.4).

Рис. 15.4. Плата расширения с жидкокристаллическим индикатором и клавиатурой от компании Freetronics

ПРИМЕЧАНИЕ

В наших примерах будет использоваться плата расширения с жидкокристаллическим индикатором и клавиатурой от компании Freetronics. За дополнительной информацией обращайтесь по адресу http://www.freetronics.com.au/collections/display/products/lcd-keypad-shield/. Если вы решите использовать другой модуль отображения, не забудьте подставить соответствующие значения в функцию LiquidCrystal.

Для отображения на экране ЖКИ текущих координат, получаемых от приемника GPS, сконструируем очень простое переносное GPS-устройство, которое будет питаться от батареи напряжением 9 В.

Оборудование

Для создания устройства понадобится немного оборудования:

- плата Arduino и кабель USB;
- модуль ЖКИ или плата расширения с жидкокристаллическим индикатором от Freetronics (упоминается выше);
- одна батарея напряжением 9 В и кабель с разъемом;
- один модуль GPS или плата расширения GPS для Arduino.

Скетч

Введите и загрузите следующий скетч:

```
  // Проект 43 – простой приемник GPS
❶ #include <LiquidCrystal.h>
  LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );

  #include <TinyGPS.h>
  #include <SoftwareSerial.h>

  TinyGPS gps;

  byte gpsData;

  // Функция getgps отображает полученные данные на экране ЖКИ
❷ void getgps(TinyGPS &gps)
  {
    float latitude, longitude;

    // Извлечь и отобразить координаты
❸   gps.f_get_position(&latitude, &longitude);
    lcd.setCursor(0, 0);
    lcd.print("Lat:");
```

```
    lcd.print(latitude, 5);
    lcd.print(" ");
    lcd.setCursor(0, 1);
    lcd.print("Long:");
    lcd.print(longitude, 5);
    lcd.print(" ");
    delay(3000); // Ждать 3 секунды
    lcd.clear();
  }

  void setup()
  {
    Serial2.begin(9600);
  }

  void loop()
  {
    while (Serial2.available() > 0)
    {
      // Получить байт данных от платы GPS
      gpsData = Serial2.read();
      if (gps.encode(gpsData))
      {
❹         getgps(gps);
      }
    }
  }
```

С ❶ по ❷ строку скетч подключает важные библиотеки для работы с жидкокристаллическим индикатором и GPS. В `void loop` символы, полученные от приемника GPS, передаются функции `getgps()` ❹. С помощью `gps.f_get_position()` ❸ она извлекает координаты в переменные `&latitude` и `&longitude` для вывода на экран ЖКИ.

Отображение координат на экране ЖКИ

Когда скетч будет загружен, и приемник GPS начнет принимать данные, на экране ЖКИ появятся широта и долгота вашего местоположения, как на рис. 15.5.

Рис. 15.5. Широта и долгота, отображаемые скетчем из проекта 43

Но где находится эта точка на поверхности Земли? Это можно узнать с помощью Google Maps (http://maps.google.com/). Откройте страницу Google Maps в браузере, введите в поле поиска широту и долготу, разделив их запятой и пробелом, и вы увидите карту с отмеченной на ней точкой. Например, для координат на рис. 15.5 Google Maps выведет такую карту, как на рис. 15.6.

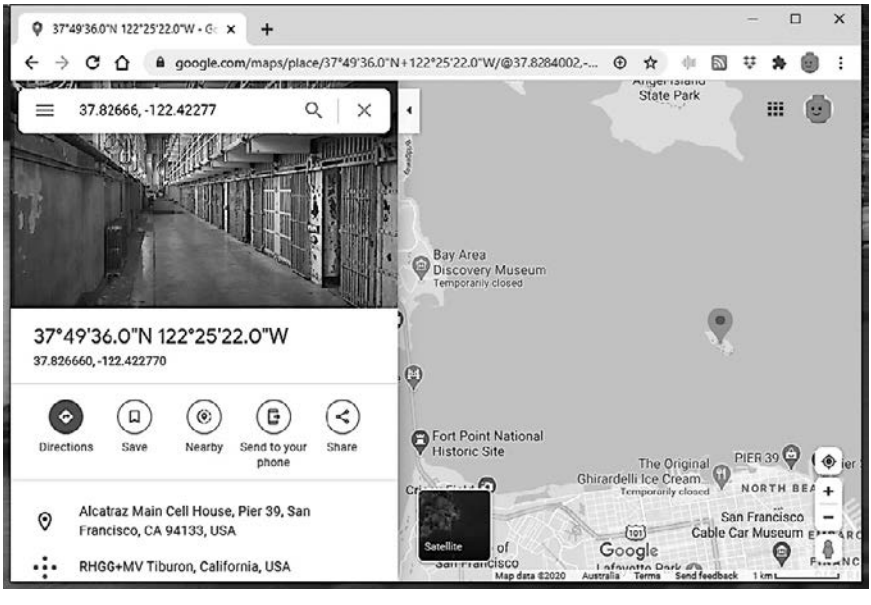


Рис. 15.6. Координаты на рис. 15.5 — это координаты острова Алькатрас

Проект 44: часы точного времени на основе GPS

С помощью GPS можно не только определять место, но и получать информацию о времени, на основе которой можно создать очень точные часы.

Оборудование

Здесь мы используем оборудование из проекта 43.

Скетч

Введите следующий скетч, извлекающий время из данных GPS:

```
// Проект 44 — часы для отображения точного времени на основе GPS
#include <LiquidCrystal.h>
```

```
#include <TinyGPS.h>
#include <SoftwareSerial.h>

LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );
// GPS RX к D3, GPS TX к D2

SoftwareSerial Serial2(2, 3);

TinyGPS gps;

byte gpsData;

void getgps(TinyGPS &gps)
{
  int year,a,t;
  byte month, day, hour, minute, second, hundredths;
❶ gps.crack_datetime(&year,&month,&day,
                     &hour,&minute,&second,&hundredths);
❷ hour=hour+10; // Исправьте для вашего часового пояса
  if (hour>23)
  {
    hour=hour-24;
  }
  lcd.setCursor(0,0); // Вывести дату и время
❸ lcd.print("Current time: ");
  lcd.setCursor(4,1);
  if (hour<10)
  {
    lcd.print("0");
  }
  lcd.print(hour, DEC);
  lcd.print(":");
  if (minute<10)
  {
    lcd.print("0");
  }
  lcd.print(minute, DEC);
  lcd.print(":");
  if (second<10)
  {
    lcd.print("0");
  }
  lcd.print(second, DEC);
}

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

```
}

void loop()
{
  while (Serial2.available() > 0)
  {
    // Получить байт данных от платы GPS
    gpsData = Serial2.read();
    if (gps.encode(gpsData))
    {
      getgps(gps);
    }
  }
}
```

Этот скетч работает так же, как и тот, что был в проекте 43, только извлекает время, а не координаты. Это всегда время по Гринвичу (Всемирное координированное время) ❶. В строке ❷ вы можете добавить либо вычесть нужное количество часов, чтобы привести время к своему часовому поясу. Полученные данные форматируются и отображаются на экране ЖКИ ❸. Пример вывода времени показан на рис. 15.7.

Рис. 15.7. Проект 44 во время работы

Проект 45: запись координат перемещающегося объекта с течением времени

Вы знаете, как принимать координаты GPS и преобразовывать их в нормальный числовой вид. Теперь создадим GPS-трекер и организуем запись этой информации в плату расширения с картой памяти microSD из главы 7. Наш трекер будет определять и сохранять свои координаты. Благодаря использованию карты памяти microSD можно записать траекторию движения автомобиля, катера и другого перемещающегося объекта, с которого возможен прием сигналов GPS, а потом просмотреть эту информацию на компьютере.

Оборудование

В этом проекте будет использоваться оборудование из предыдущих примеров. Разница лишь в том, что из устройства нужно убрать плату расширения с жидкокристаллическим индикатором. Если вы используете отдельный модуль GPS, то понадобится плата расширения с картой памяти SD. Еще вам нужен внешний источник питания. В этом примере будут сохраняться время, координаты и примерная скорость движения.

Скетч

После сборки устройства введите и загрузите следующий скетч:

```
// Проект 45 — запись координат перемещающегося объекта
//             с течением времени

#include <TinyGPS.h>
#include <SoftwareSerial.h>
#include <SD.h>

// GPS TX to D2, RX to D3

SoftwareSerial Serial2(2, 3);

TinyGPS gps;

byte gpsData;

void getgps(TinyGPS &gps)
{
  float latitude, longitude;
  int year;
  byte month, day, hour, minute, second, hundredths;

  // Создать/открыть файл для записи
  File dataFile = SD.open("DATA.TXT", FILE_WRITE);
  // Если файл готов, записать в него данные
❶ if (dataFile)
  {
❷   gps.f_get_position(&latitude, &longitude);
    dataFile.print("Lat: ");
    dataFile.print(latitude,5);
    dataFile.print(" ");
    dataFile.print("Long: ");
    dataFile.print(longitude,5);
    dataFile.print(" ");
```

```
    // Декодировать и записать время
    gps.crack_datetime(&year,&month,&day,
                       &hour,&minute,&second,&hundredths);

    // Исправьте для вашего часового пояса,
    // как в проекте 44
    hour=hour+10;
    if (hour>23)
    {
      hour=hour-24;
    }
    if (hour<10)
    {
      dataFile.print("0");
    }
    dataFile.print(hour, DEC);
    dataFile.print(":");
    if (minute<10)
    {
      dataFile.print("0");
    }
    dataFile.print(minute, DEC);
    dataFile.print(":");
    if (second<10)
    {
      dataFile.print("0");
    }
    dataFile.print(second, DEC);
    dataFile.print(" ");
    dataFile.print(gps.f_speed_kmph());
❸   dataFile.println("km/h");
    dataFile.close();
❹   delay(15000); // Записывать координаты каждые 15 секунд
  }
// Если файл не готов, показать ошибку:
  else
  {
    Serial.println("error opening DATA.TXT");
  }
}

void setup()
{
  Serial.begin(9600);
  Serial2.begin(9600);
  Serial.println("Initializing SD card...");
  pinMode(10, OUTPUT);
```

```
  // Проверить наличие карты памяти microSD
  // и возможность ее использования
  if (!SD.begin(10)) {
    Serial.println("Card failed, or not present");
    // Остановить скетч
    return;
  }
  Serial.println("memory card is ready");
}

void loop()
{
  while (Serial2.available() > 0)
  {
    // Получить байт данных от платы GPS
    gpsData = Serial2.read();
    if (gps.encode(gpsData))
    {
❺     getgps(gps);
    }
  }
}
```

Этот скетч использует в функции `void loop()` код из проектов 43 и 44, получающий данные из приемника GPS и передающий их другим функциям. В строке ❺ полученные из приемника данные передаются библиотеке TinyGPS для декодирования в нормальное числовое представление. В строке ❶ проверяется наличие карты памяти и возможность записи на нее. Со ❷ по ❸ строки извлеченные данные GPS записываются в текстовый файл на карте microSD. После каждой записи файл закрывается, поэтому вы можете отключить питание Arduino в любой момент, не опасаясь потерять данные. Более того, вы должны это делать перед вставкой или извлечением microSD. В строке ❹ можно установить интервал между записями, изменив значение аргумента в вызове функции `delay()`.

Отображение траектории на карте

После опробования GPS-логгера содержимое получившегося текстового файла должно выглядеть как на рис. 15.8.

Если полученные данные вручную ввести в Google Maps, можно получить схему движения на карте. Но гораздо интереснее наложить на карту сразу всю траекторию. Для этого откройте текстовый файл в электронной таблице, отделите координаты и добавьте строку заголовка, как на рис. 15.9. После сохраните результат в файле с расширением `.csv`.

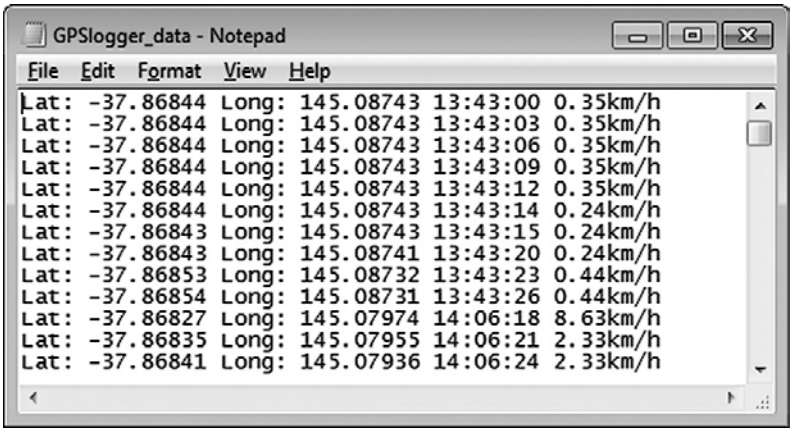


Рис. 15.8. Результаты работы проекта 45

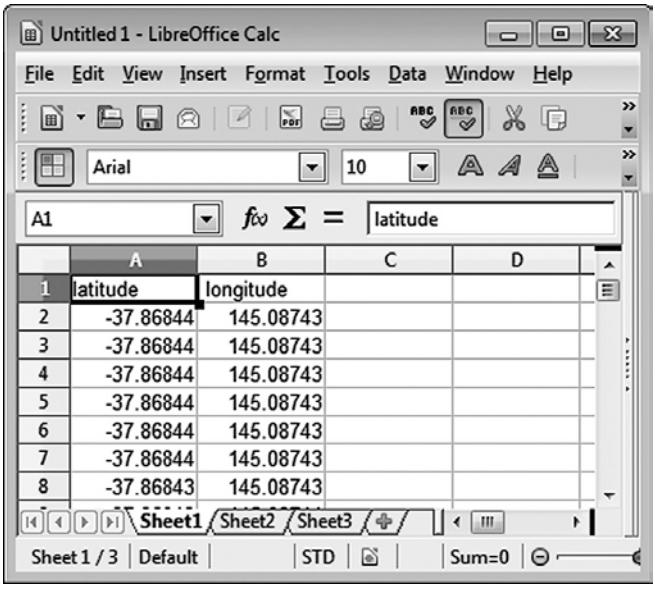


Рис. 15.9. Извлеченные координаты

Теперь откройте в браузере сайт GPS Visualizer (http://www.gpsvisualizer.com/). В разделе Get Started Now (Приступить прямо сейчас) нажмите Choose File (Выбрать файл) и выберите свой файл с данными. В поле Choose an output format (Выходной формат) выберите пункт Google Maps и нажмите Map It (Отобразить). Данные,

зафиксированные вашим GPS-трекером, должны отобразиться на карте в виде траектории (рис. 15.10). Затем ее можно отмасштабировать и исследовать.

Рис. 15.10. Траектория, извлеченная из GPS-трекера и наложенная на карту

Что дальше?

Как видите, даже то, что раньше казалось слишком сложным, например работа с приемниками GPS, на деле оказывается простым благодаря Arduino. Продолжим эту тему в следующей главе. Вы узнаете, как создать свой беспроводной канал передачи данных и как организовать дистанционное управление.

16

Беспроводная передача информации

В этой главе вы:

- узнаете, как посылать и принимать инструкции и данные с помощью разного оборудования для беспроводной связи;
- научитесь посылать цифровые сигналы с помощью недорогих модулей беспроводной связи;
- создадите простую и бюджетную систему дистанционного управления;
- узнаете, как пользоваться беспроводными приемопередатчиками LoRa;
- создадите термометр на дистанционном управлении.

Применение недорогих модулей беспроводной связи

Текстовую информацию в одном направлении легко можно передать с помощью беспроводного канала, связывающего две системы на основе Arduino с недорогими модулями, работающими в радиочастотном диапазоне, такими как радиопередатчик и приемник на рис. 16.1. Обычно эти модули продаются парами, и их часто называют модулями или наборами *RF Link* (радиосвязи). Прекрасные примеры — модуль 44910433 компании PMD Way или модули WRL-10534 и WRL-10532 компании SparkFun. В наших проектах мы будем использовать самые популярные типы модулей, работающие в радиодиапазоне 433 МГц.

Выводы в нижней части модуля передатчика на рис. 16.2 имеют следующие назначения *(слева направо)*: вход данных, питание 5 В и «земля». Контакт для подключения

внешней антенны находится в правом верхнем углу платы. Антенной может служить кусок провода. Но обойтись можно и без нее, если передача происходит на короткие расстояния (конструкция разных моделей может немного отличаться. Обязательно загляните в документацию, чтобы определить назначение выводов модуля, прежде чем выполнять его подключение).

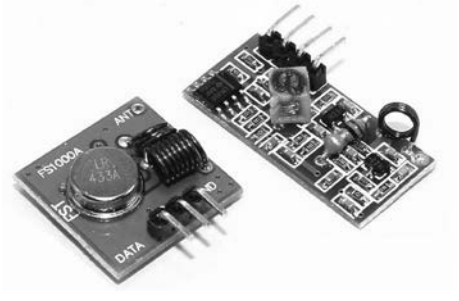


Рис. 16.1. Модули передатчика и приемника в радиодиапазоне

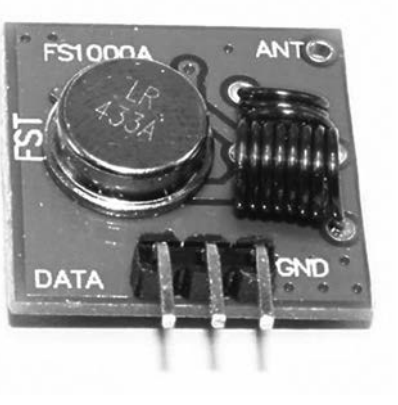


Рис. 16.2. Модуль передатчика в радиодиапазоне

На рис. 16.3 показан модуль приемника, он немного крупнее модуля передатчика.

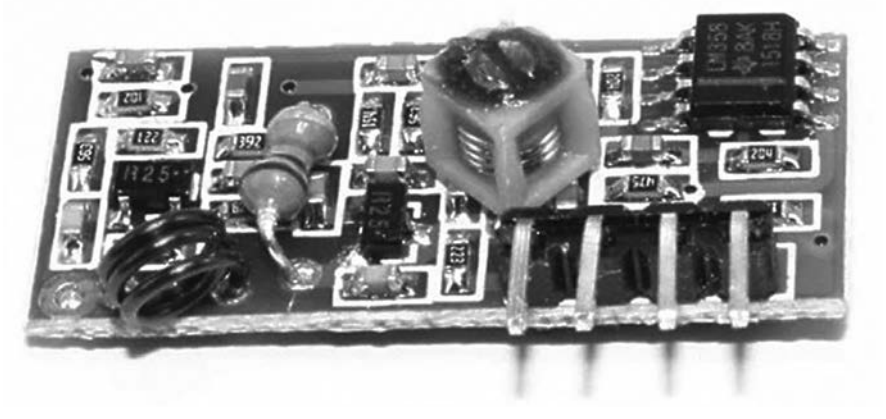


Рис. 16.3. Модуль приемника в радиодиапазоне

Подключается приемник просто: контакты V+ и V– подключите к контактам 5 V и GND, а контакт DATA приемника — к контакту для приема данных на плате Arduino.

Маркировка этих контактов обычно наносится с обратной стороны модуля. Если с прочтением маркировки возникли трудности, поищите описание модуля в интернете или обратитесь к продавцу.

До начала работы с модулями загрузите и установите последнюю версию библиотеки VirtualWire: http://www.airspayce.com/mikem/arduino/VirtualWire/, как описывалось в главе 7. Вместе с библиотекой в архиве с примерами для этой книги есть и скетч. Архив доступен по адресу https://nostarch.com/arduino-workshop-2nd-edition/. После установки можно переходить к следующему разделу.

ПРИМЕЧАНИЕ

Модули радиосвязи бюджетны и просты в использовании, но в них нет средств проверки и исправления ошибок, которые гарантировали бы достоверность принимаемых данных. Поэтому лучше использовать их только для решения простых задач (например, для несложного дистанционного управления). Если вам нужна высокая надежность передачи данных, используйте модули LoRa и подобные им. О них рассказывается дальше.

Проект 46: пульт дистанционного управления

В этом проекте мы реализуем дистанционное управление двумя цифровыми выходами. Вы будете нажимать кнопки, подключенные к одной плате Arduino, и с их помощью управлять соответствующими цифровыми выходами на другой, расположенной неподалеку. Этот проект наглядно покажет, как пользоваться модулями радиосвязи и как определить максимальное расстояние, на котором еще возможно дистанционное управление с применением модулей. Дальше вы уже сможете использовать их для решения более сложных задач (на открытом воздухе максимальное расстояние — примерно 100 м, но в закрытых помещениях оно будет меньше из-за поглощения радиоволн препятствиями между модулями).

Оборудование для передатчика

Для сборки передатчика нужно следующее оборудование:

- плата Arduino и кабель USB;
- контейнер для батарей AA и кабель с разъемом (использовались в главе 14);
- один модуль передатчика, работающий в радиодиапазоне 433 МГц;
- два резистора номиналом 10 кОм (R1 и R2);
- два конденсатора емкостью 100 нФ (C1 и C2);
- две кнопки без фиксации;
- одна макетная плата.

Схема передатчика

На схеме передатчика есть две кнопки без фиксации с фильтром против дребезга контактов, подключенные к цифровым контактам 2 и 3, и модуль передатчика, подключенный, как описывалось выше (рис. 16.4).

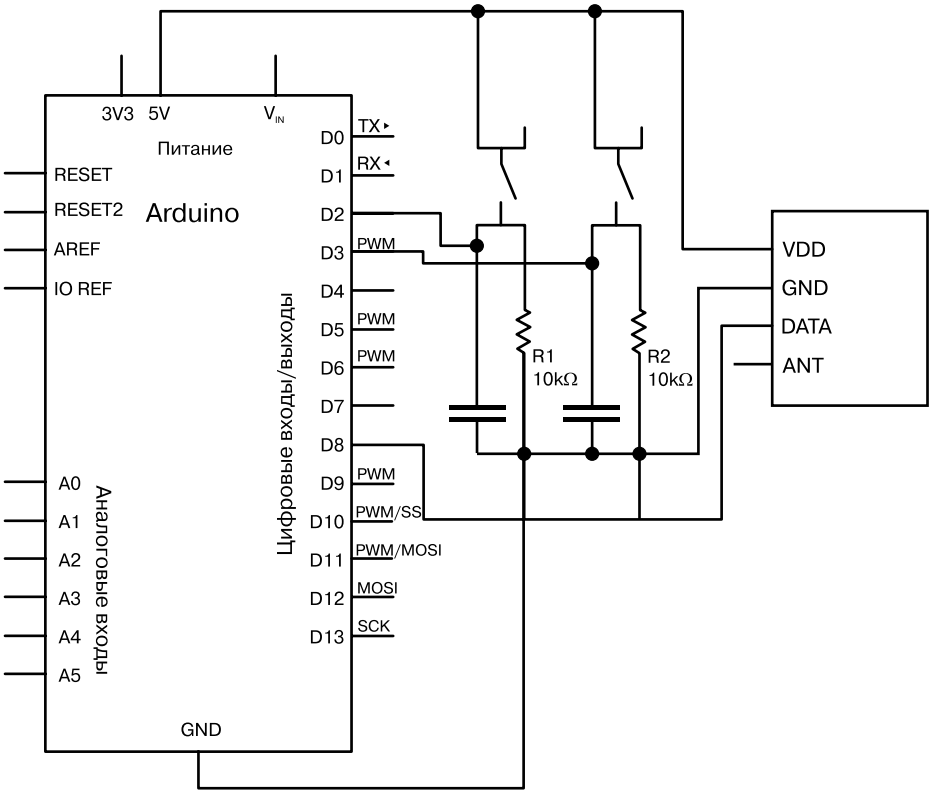


Рис. 16.4. Схема передатчика для проекта 46

Оборудование для приемника

Для сборки приемника нужно следующее оборудование:

- плата Arduino и кабель USB;
- контейнер для батарей AA и кабель с разъемом (использовались в главе 14);
- один модуль приемника, работающий в радиодиапазоне 433 МГц;

- одна макетная плата;
- два светодиода по вашему выбору;
- два резистора номиналом 560 Ом (R1 и R2).

Схема приемника

Схема приемника содержит два светодиода, подключенных к цифровым контактам 6 и 7. Контакт DATA модуля приемника подключен к цифровому контакту 8, как на рис. 16.5.

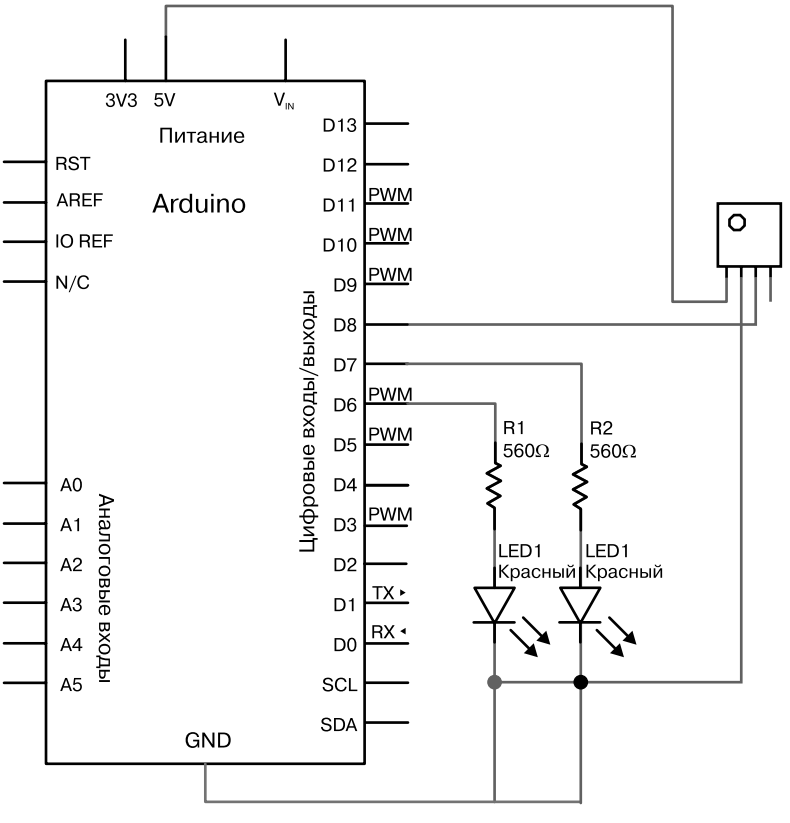


Рис. 16.5. Схема приемника для проекта 46

Вы можете заменить макетную плату, светодиоды, резисторы и модуль приемника экраном приемника Freetronics 433 МГц от компании Freetronics (рис. 16.6).

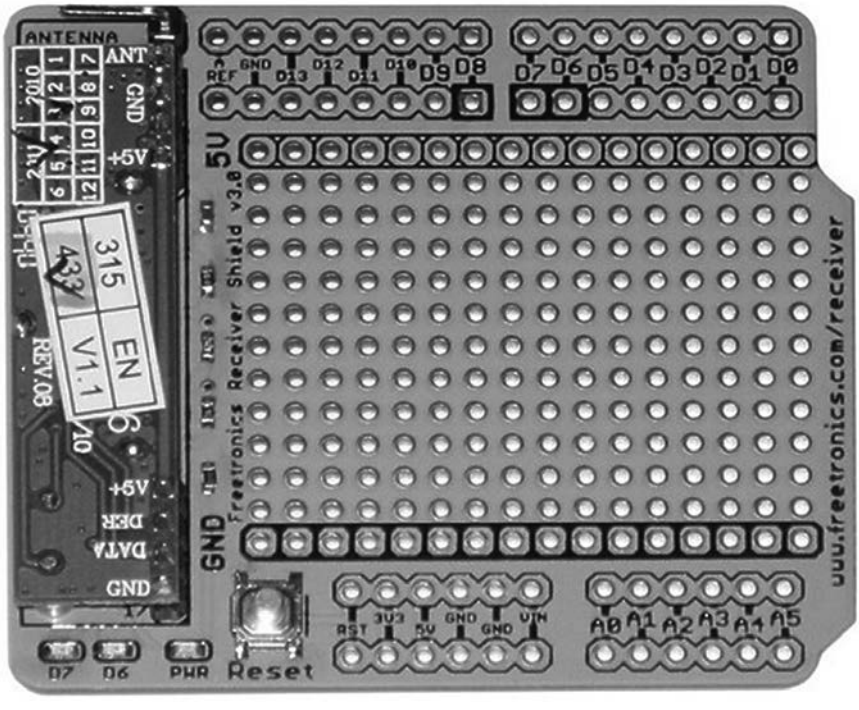


Рис. 16.6. Плата расширения с модулем приемника для радиодиапазона 433 МГц от компании Freetronics

Скетч передатчика

Теперь рассмотрим скетч передатчика. Введите и загрузите следующий скетч в плату Arduino с подключенным модулем передатчика:

```
// Проект 46 – пульт дистанционного управления.
//             Скетч передатчика
❶ #include <VirtualWire.h>

uint8_t buf[VW_MAX_MESSAGE_LEN];
uint8_t buflen = VW_MAX_MESSAGE_LEN;

❷ const char *on2 = "a";
const char *off2 = "b";
const char *on3 = "c";
const char *off3 = "d";

void setup()
{
```

```
❸   vw_set_ptt_inverted(true); // Для модуля передатчика необходимо
    vw_setup(300);             // установить скорость передачи
❹   vw_set_tx_pin(8);
    pinMode(2, INPUT);
    pinMode(3, INPUT);
  }

  void loop()
  {
❺   if (digitalRead(2)==HIGH)
    {
      vw_send((uint8_t *)on2, strlen(on2)); // Отправить данные
      vw_wait_tx();                         // Подождать немного
      delay(200);
    }
    if (digitalRead(2)==LOW)
    {
❻     vw_send((uint8_t *)off2, strlen(off2));
      vw_wait_tx();
      delay(200);
    }
    if (digitalRead(3)==HIGH)
    {
      vw_send((uint8_t *)on3, strlen(on3));
      vw_wait_tx();
      delay(200);
    }
    if (digitalRead(3)==LOW)
    {
      vw_send((uint8_t *)off3, strlen(off3));
      vw_wait_tx();
      delay(200);
    }
  }
```

Работа с модулями радиосвязи в скетчах осуществляется функциями ❸ из библиотеки VirtualWire ❶. В ❹ выбирается цифровой вывод 8, который будет использоваться для обмена данными между Arduino и модулем передатчика, и устанавливается скорость передачи (вы можете использовать любой другой цифровой вывод, кроме 0 и 1, которые соединены с последовательным портом).

Скетч передатчика читает состояния двух кнопок, подключенных к цифровым контактам 2 и 3, и посылает в модуль радиосвязи один текстовый символ, соответствующий этим состояниям. Например, когда нажимается кнопка, подключенная к цифровому контакту 2, на нем устанавливается уровень `HIGH` и Arduino посылает символ *a*, а когда кнопка отпускается — *b*. Когда нажимается кнопка, подключенная к цифровому контакту 3, на нем устанавливается уровень `HIGH` и Arduino посылает символ *c*, а когда кнопка отпускается — *d*. Все четыре возможных состояния определяются начиная со строки ❷.

Передача символа выполняется в одной из четырех инструкций `if`, начиная с ❺. К примеру, в инструкции `if-then` ❻. Переменная для передачи используется дважды — например, `on2`, как показано ниже:

```
vw_send((uint8_t *)on2, strlen(on2));
```

Функция `vw_send` посылает содержимое переменной `on2`, но ей нужно знать размер этого содержимого в символах. Он определяется вызовом функции `strlen()`.

Скетч приемника

Теперь добавим скетч приемника. Введите и загрузите следующий скетч в Arduino с подключенным модулем приемника:

```
// Проект 46 — пульт дистанционного управления.
//             Скетч приемника

#include <VirtualWire.h>

uint8_t buf[VW_MAX_MESSAGE_LEN];
uint8_t buflen = VW_MAX_MESSAGE_LEN;

void setup()
{
❶  vw_set_ptt_inverted(true); // Необходимо для модуля приемника
   vw_setup(300);
❷  vw_set_rx_pin(8);
   vw_rx_start();
   pinMode(6, OUTPUT);
   pinMode(7, OUTPUT);
}

void loop()
{
❸  if (vw_get_message(buf, &buflen))
   {
❹    switch(buf[0])
     {
       case 'a':
         digitalWrite(6, HIGH);
         break;
       case 'b':
         digitalWrite(6, LOW);
         break;
       case 'c':
         digitalWrite(7, HIGH);
         break;
       case 'd':
         digitalWrite(7, LOW);
         break;
     }
   }
}
```

Как и в скетче передатчика, здесь используются функции из библиотеки VirtualWire для настройки модуля приемника ❶ и установки скорости приема данных. В ❷ настраивается цифровой вывод на плате Arduino, через который будут приниматься входящие данные (контакт 8).

Во время работы скетча модуль приемника принимает символы от передатчика и посылает в Arduino. Вызовом функции `vw_get_message()` ❸ скетч извлекает символы, полученные платой, и интерпретирует их с помощью инструкции `switch-case` ❹. Например, нажатие кнопки S1 на передатчике вызовет отправку символа *a*. После получения символа приемник установит уровень `HIGH` на цифровом выходе 6 и включит светодиод.

Эту пару простых устройств можно использовать для более сложного алгоритма дистанционного управления системами на основе Arduino, посылающего коды в виде простых символов, которые будут интерпретироваться на стороне приемника.

Использование модулей LoRa для быстрой беспроводной передачи данных на большие расстояния

Чтобы организовать беспроводную передачу данных на большие расстояния и со скоростью выше, чем у простых модулей радиосвязи, можно использовать модули LoRa. LoRa — это сокращение от long range («с большим радиусом действия»). Модули этого типа способны устанавливать связь на больших расстояниях и потребляют мало энергии. Они относятся к категории *приемопередатчиков* (трансиверов) — устройств для приема и передачи данных. С ними нам не нужно использовать разные модули передатчика и приемника. Еще одно преимущество модулей LoRa — способность обмениваться данными с разными типами модулей. Это позволит вам как разработчику создавать сети управления и передачи данных от простых до сложных. В этой главе вы создадите несколько простых устройств, которые можно использовать для разных целей.

Для удобства мы будем использовать две платы расширения LoRa для Arduino. Например, выпускаемые компанией PMD Way с артикулом 14290433. Одну из них можно увидеть на рис. 16.7.

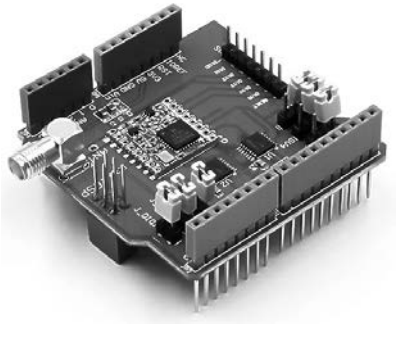


Рис. 16.7. Плата расширения LoRa для Arduino

При использовании плат расширения LoRa вам придется выбрать рабочую частоту. Она зависит от страны. Это нужно, чтобы ваши

конструкции не мешали работе других устройств. Продукты LoRa выпускаются в таких рабочих частотных диапазонах:

- **433 МГц** — для США и Канады;
- **864–865,5; 868,7–869,2 МГц** — для России;
- **868 МГц** — для Великобритании и Европы;
- **915 МГц** — для Австралии и Новой Зеландии.

Полный список стран и разрешенных диапазонов радиочастот в каждой из них можно найти по ссылке https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country.html.

Теперь загрузите и установите библиотеку для Arduino, которую можно найти по адресу https://github.com/sandeepmistry/arduino-LoRa/archive/master.zip.

Проект 47: беспроводная передача данных с помощью LoRa

Этот проект продемонстрирует простую передачу данных от одного Arduino, оснащенного LoRA, к другому, чтобы обеспечить дистанционное управление цифровыми выходами. На передатчике есть две кнопки. Их нажатием можно управлять цифровыми выходами на приемнике.

Оборудование для передатчика

Для сборки передатчика нужно следующее оборудование:

- плата Arduino и кабель USB;
- плата расширения LoRa;
- два резистора номиналом 10 кОм (R1 и R2);
- два конденсатора емкостью 100 нФ (C1 и C2);
- две кнопки без фиксации;
- контейнер для батарей AA и кабель с разъемом (использовались в главе 14).

Схема передатчика

Схема передатчика (рис. 16.8) содержит две кнопки без фиксации с фильтром против дребезга контактов, подключенные к цифровым контактам 2 и 3. Плата расширения LoRa вставляется контактами в разъемы на Arduino Uno. После загрузки скетча питание конструкции будет от батарей AA. Перед использованием платы расширения LoRa нужно убрать три перемычки, показанные на рис. 16.9, иначе они будут мешать работе других цифровых контактов. Вы можете убрать их полностью или просто подключить каждую только к одному из двух контактов.

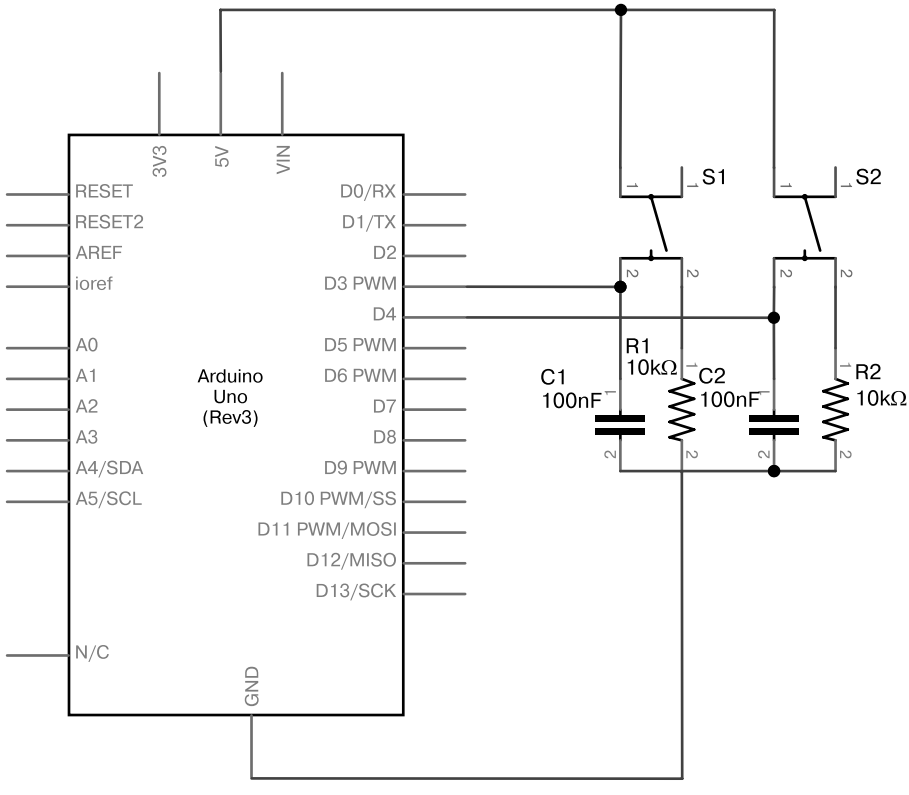


Рис. 16.8. Схема передатчика для проекта 47

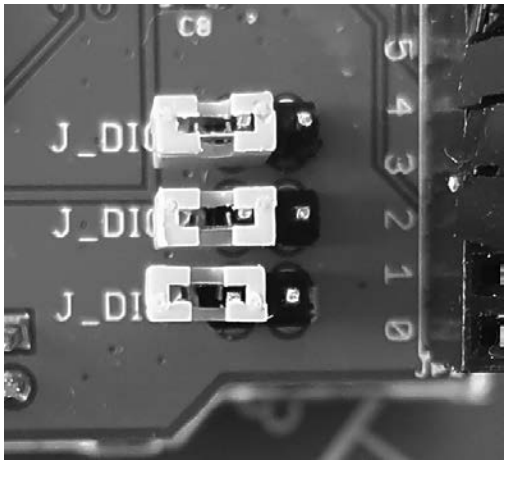


Рис. 16.9. Перемычки на плате расширения LoRa, которые необходимо убрать

Оборудование для приемника

Для сборки приемника нужно следующее оборудование:

- плата Arduino и кабель USB;
- плата расширения LoRa;
- один светодиод;
- один резистор номиналом 560 Ом (R1).

Схема приемника

Схема приемника на рис. 16.10 содержит один светодиод и ограничительный резистор, включенный между цифровым выводом 7 и светодиодом. Приемник в этом проекте останется подключенным к компьютеру через USB, поэтому внешний источник питания не нужен.

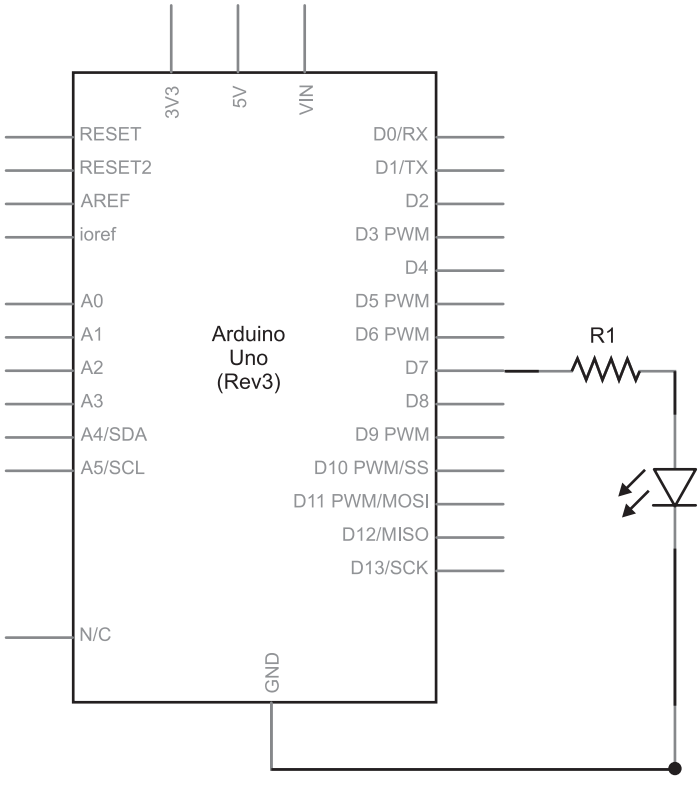


Рис. 16.10. Схема приемника для проекта 47

Скетч передатчика

Теперь рассмотрим скетч передатчика. Введите и загрузите следующий скетч в плату Arduino, которая будет играть роль передатчика:

```
// Проект 47 – дистанционное управление через LoRa, скетч передатчика
❶ #define LORAFREQ (915000000L)
❷ #include <LoRa.h>
  #include <SPI.h>
❸ void loraSend(int controlCode)
  {
❹   LoRa.beginPacket();      // Начать отправку данных
    LoRa.print("ABC");       // ABC – наш трехсимвольный код для приемника
    LoRa.print(controlCode); // Послать управляющий код (в controlCode)
❺   LoRa.endPacket();        // Завершить отправку данных
❺   LoRa.receive();          // Начать прием
  }

  void setup()
  {
    pinMode(4, INPUT);    // Кнопка "включить"
    pinMode(3, INPUT);    // Кнопка "выключить"
❻   LoRa.begin(LORAFREQ); // Настроить частоту в LoRa
  }

  void loop()
  {
    // Проверить нажатие кнопки управления приемником
    if (digitalRead(4) == HIGH)
    {
      loraSend(1); // '1' – код для подачи уровня HIGH на цифровой вывод 5
      delay(500);  // Дать время на отправку команды
    }
    if (digitalRead(3) == HIGH)
    {
      loraSend(0); // '0' – код для подачи уровня LOW на цифровой вывод 5
      delay(500);  // Дать время на отправку команды

    }
}
```

В строке ❶ определяется рабочая частота. Здесь используется частота 915 МГц. Если в вашей стране этот диапазон запрещен для использования, придется заменить его на `433000000L` или `868000000L`, в зависимости от страны и платы расширения. Библиотека поддержки LoRa подключается в ❷ и активируется в ❹. К скетчу подключается и библиотека SPI (плата расширения LoRa взаимодействует с Arduino через интерфейс SPI). В строке ❻ после настройки цифровых входов для кнопок скетч активирует передатчик LoRa и настраивает его на использование соответствующей частоты.

В ❸ определяется функция `loraSend(int controlCode)`. Она вызывается при нажатии кнопки. Первым делом эта функция посылает в эфир трехзначный код, в нашем случае `ABC`, а после — управляющий код. Трехзначный код позволяет отправлять команды конкретному приемнику. Если одновременно будут работать несколько приемников, они все будут реагировать на управляющий код, что может привести к путанице. Дальше вы увидите, что приемник будет выполнять команды, только если перед этим получено сообщение `ABC`. Управляющие коды в нашем примере — это цифры `1` и `0` (они включают или выключают цифровой выход на стороне приемника).

В ❹ модуль LoRa переключается в режим передачи. Затем ему передаются трехзначный и управляющий коды для отправки в радиоэфир. В ❺ модулю LoRa отдается команда прекратить передачу и переключиться на прием данных. После загрузки эскиза передатчика его аппаратное обеспечение можно отключить от компьютера и включить питание от аккумуляторной батареи.

Скетч приемника

Теперь рассмотрим скетч для приемника. Введите и загрузите следующий скетч в плату Arduino, которая будет играть роль приемника:

```
// Проект 47 — дистанционное управление через LoRa, скетч приемника
❶ #define LORAFREQ (915000000L)
❷ #include <LoRa.h>
  #include <SPI.h>

  // Определяет, как обрабатываются данные, принятые платой расширения LoRa
  void takeAction(int packetSize)
  {
❸   char incoming[4] = "";
    int k;
    for (int i = 0; i < packetSize; i++)
    {
      k = i;
      if (k > 6)
      {
        k = 6; // Гарантировать невозможность записи за пределы строки
      }
      incoming[k] = (char)LoRa.read();
❹   }
    // Проверить трехзначный код, отправленный передатчиком
❺   if (incoming[0] != 'A')
    {
      return; // Если не 'A', прервать обработку и вернуться в void loop()
    }
```

```
❺   if (incoming[1] != 'B')
    {
      return; // Если не 'B', прервать обработку и вернуться в void loop()
    }
❺   if (incoming[2] != 'C')
    {
      return; // Если не 'C', прервать обработку и вернуться в void loop()
    }

    // Добравшись до этой точки, мы можем быть уверены в верности
    // трехзначного кода, полученного от передатчика.
    // Теперь можно обработать управляющий код...
    if (incoming[3] == '1')
    {
      digitalWrite(7, HIGH);
    }
    if (incoming[3] == '0')
    {
      digitalWrite(7, LOW);
    }
  }

  void setup()
  {
    pinMode(7, OUTPUT);
❻   LoRa.begin(LORAFREQ);       // Настроить частоту в LoRa
❼   LoRa.onReceive(takeAction); // Вызывать takeAction при получении данных
❽   LoRa.receive();             // Начать прием
  }

  void loop()
  {
  }
```

Мы снова подключаем ❷ и активируем ❻ библиотеку поддержки LoRa. В строке ❶ определяется рабочая частота. В этом примере используется частота 915 МГц. Если в вашей стране этот диапазон запрещен для использования, придется заменить его на `433000000L` или `868000000L`, в зависимости от страны и платы расширения. К скетчу подключается и библиотека SPI, потому что плата расширения LoRa взаимодействует с Arduino через интерфейс SPI. В строке ❼ мы задаем функцию — в нашем случае `void takeAction()`, — которая должна вызываться при приеме символов из радиоэфира. Затем в ❽ модуль LoRa переключается в режим приема.

В процессе работы приемник просто ждет получения данных модулем LoRa. Когда будут получены любые данные, библиотека вызовет функцию `takeAction()`. Эта функция принимает символы от передатчика и помещает их в массив `incoming[4]`

в строках с ❸ по ❹. Затем символы проверяются на принадлежность трехзначному коду (в нашем случае `ABC`), чтобы убедиться, что полученные данные предназначены для этого приемника ❺. Если все прошло успешно, проверяется управляющий символ. Если это `1`, то на цифровом выводе `7` устанавливается уровень `HIGH`, а если это `0` — `LOW`.

Теперь у вас есть основа для создания устройств с дистанционным управлением на больших расстояниях. Назначив разные трехзначные коды нескольким приемникам, можно расширить систему и организовать управление несколькими принимающими устройствами с помощью одного передатчика.

Но при решении серьезных задач может понадобиться получить от приемника подтверждение выполнения команды. Поэтому добавим функцию подтверждения в следующем проекте.

Проект 48: беспроводная передача данных с подтверждением

Этот проект добавляет подтверждения в систему «приемник — передатчик» из проекта 47, организуя двустороннюю передачу данных. Светодиод в конструкции передатчика включается, когда на цифровой выход приемника подается уровень `HIGH`, и гаснет, когда подается уровень `LOW`.

Оборудование для передатчика

Для сборки передатчика нужно следующее оборудование:

- плата Arduino и кабель USB;
- плата расширения LoRa;
- два резистора номиналом 10 кОм (R1 и R2);
- один светодиод;
- два конденсатора емкостью 100 нФ (C1 и C2);
- две кнопки без фиксации;
- контейнер для батарей AA и кабель с разъемом (использовались в главе 14).

Схема передатчика

В схеме передатчика (рис. 16.11) есть две кнопки без фиксации с фильтром против дребезга контактов, подключенные к цифровым выводам `3` и `4`, один светодиод и ограничительный резистор, включенный между цифровым выводом `6` и светодиодом. Плата расширения LoRa вставляется контактами в разъемы на Arduino Uno. После загрузки скетча питание конструкции будет от батарей AA.

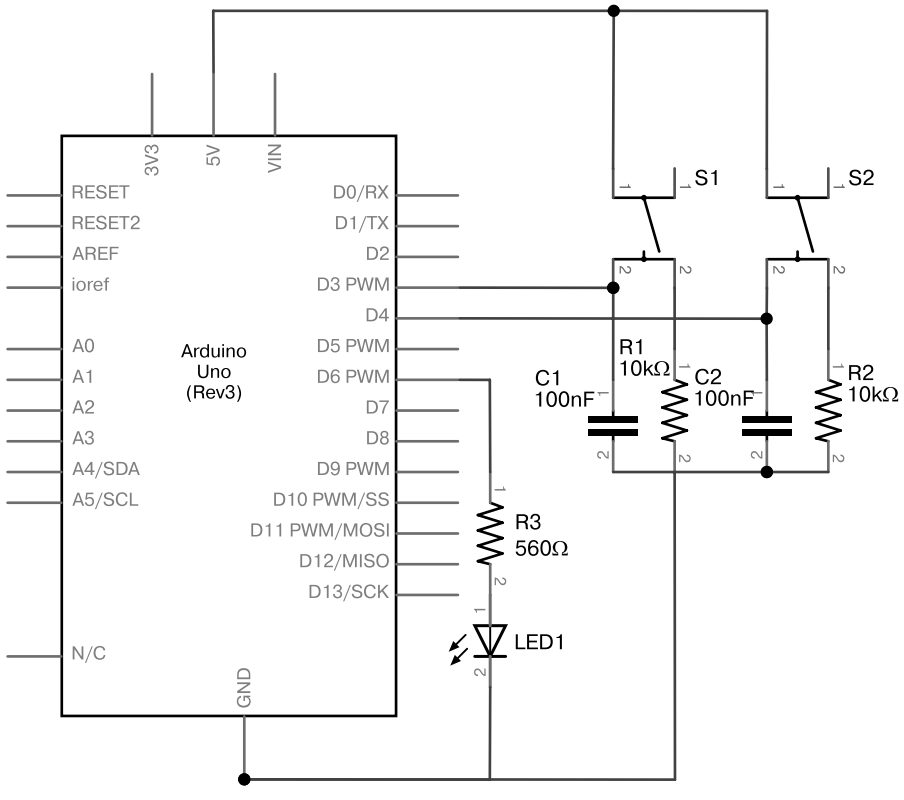


Рис. 16.11. Схема передатчика для проекта 48

Оборудование и схема приемника в этом проекте такие же, как в проекте 47.

Скетч передатчика

Теперь рассмотрим скетч передатчика. Введите и загрузите следующий скетч в плату Arduino, которая будет играть роль передатчика:

```
// Проект 48 – дистанционное управление через LoRa с подтверждением
// Скетч передатчика
#define LORAFREQ (915000000L)
#include <SPI.h>
#include <LoRa.h>

void loraSend(int controlCode)
{
  LoRa.beginPacket();      // Начать отправку данных
  LoRa.print("DEF");       // DEF – наш трехсимвольный код для приемника
```

```
                              // Требуется для идентификации приемника
    LoRa.print(controlCode); // Послать управляющий код (в controlCode)
    LoRa.endPacket();        // Завершить отправку данных
    LoRa.receive();          // Начать прием
  }

  // Определяет, как обрабатываются данные, принятые платой расширения LoRa
❶ void takeAction(int packetSize)
  {
    char incoming[4] = "";
    int k;
    for (int i = 0; i < packetSize; i++)
    {
      k = i;
      if (k > 6)
      {
        k = 18; // Гарантировать невозможность записи за пределы строки
      }
      incoming[k] = (char)LoRa.read();
    }
    // Проверить трехзначный код, отправленный приемником
    if (incoming[0] != 'D')
    {
      return; // Если не 'D', прервать обработку и вернуться в void loop()
    }
    if (incoming[1] != 'E')
    {
      return; // Если не 'E', прервать обработку и вернуться в void loop()
    }
    if (incoming[2] != 'F')
    {
      return; // Если не 'F', прервать обработку и вернуться в void loop()
    }

    // Добравшись до этой точки, мы можем быть уверены в верности
    // трехзначного кода, полученного от приемника.
    // Теперь можно обработать управляющий код...
❷  if (incoming[3] == '1')
    {
      digitalWrite(6, HIGH);
      // Приемник включил свой выходной сигнал и прислал подтверждение этого
    }
❷   if (incoming[3] == '0')
    {
      digitalWrite(6, LOW);
      // Приемник выключил свой выходной сигнал и прислал подтверждение этого
    }
  }
```

```
void setup()
{
  pinMode(4, INPUT);          // Кнопка "включить"
  pinMode(3, INPUT);          // Кнопка "выключить"
  pinMode(6, OUTPUT);         // Светодиод состояния подтверждения
  LoRa.begin(LORAFREQ);       // Настроить частоту в LoRa
  LoRa.onReceive(takeAction); // Вызывать takeAction при получении данных
                              // от платы расширения LoRa
}

void loop()
{
  // Проверить нажатие кнопки управления приемником
  if (digitalRead(4) == HIGH)
  {
    loraSend(1); // '1' – код для подачи уровня HIGH на цифровой вывод 7
    delay(500);  // Защита от дребезга контактов
  }
  if (digitalRead(3) == HIGH)
  {
    loraSend(0); // '0' – код для подачи уровня LOW на цифровой вывод 7
    delay(500);  // Защита от дребезга контактов
  }
}
```

Наш передатчик работает как в проекте 47, сначала отправляя трехзначный код идентификации, а потом управляющий код для включения или выключения цифрового выхода на приемнике. Но в этом проекте передатчик дополнительно принимает сигналы от приемника. Как только приемник выполнит полученную от передатчика команду, он посылает трехзначный и управляющий код обратно передатчику.

В ❶ определяется новая для передатчика функция `takeAction()`. Она проверяет трехзначный код `DEF`, полученный от приемника. Затем приемник посылает `1`, если включил свой цифровой выход, или `0`, если выключил. После этого передатчик сообщает о полученном коде, включая или выключая свой светодиод на цифровом выводе 6 ❷.

Скетч приемника

Теперь введите и загрузите следующий скетч в плату Arduino, которая будет играть роль приемника:

```
// Проект 48 – дистанционное управление через LoRa с подтверждением
// Скетч приемника
#define LORAFREQ (915000000L)
```

```
#include <SPI.h>
#include <LoRa.h>
void loraSend(int controlCode)
{
  LoRa.beginPacket();      // Начать отправку данных
  LoRa.print("DEF");       // "DEF" — наш трехсимвольный код для передатчика

  LoRa.print(controlCode); // Послать управляющий код (в controlCode)
  LoRa.endPacket();        // Завершить отправку данных
  LoRa.receive();          // Начать прием
}
void takeAction(int packetSize)
// Определяет, как обрабатываются данные, приняты платой расширения LoRa
{
  char incoming[4] = "";
  int k;
  for (int i = 0; i < packetSize; i++)
  {
    k = i;
    if (k > 6)
    {
      k = 18; // Гарантировать невозможность записи за пределы строки
    }
    incoming[k] = (char)LoRa.read();
  }
  // Проверить трехзначный код, отправленный передатчиком
  if (incoming[0] != 'A')
  {
    return; // Если не 'A', прервать обработку и вернуться в void loop()
  }
  if (incoming[1] != 'B')
  {
    return; // Если не 'B', прервать обработку и вернуться в void loop()
  }
  if (incoming[2] != 'C')
  {
    return; // Если не 'C', прервать обработку и вернуться в void loop()
  }
  // Добравшись до этой точки, мы можем быть уверены в верности
  // трехзначного кода, полученного от приемника.
  // Теперь можно обработать управляющий код...
  if (incoming[3] == '1')
  {
    digitalWrite(7, HIGH);
  loraSend(1); // Сообщить приемнику, что выходной сигнал включен
  }
  if (incoming[3] == '0')
  {
    digitalWrite(7, LOW);
❶   loraSend(0); // Сообщить приемнику, что выходной сигнал выключен
  }
}
```

```
void setup()
{
  pinMode(7, OUTPUT);
  LoRa.begin(LORAFREQ);       // Настроить частоту в LoRa
  LoRa.onReceive(takeAction); // Вызывать takeAction при получении данных
  LoRa.receive();             // Начать прием
}
void loop()
{
}
```

Наш приемник работает как в проекте 47. Разница только в том, что он отправляет передатчику трехзначный код `DEF`, а потом цифру `1` или `0`, чтобы показать, что цифровой выход включен или выключен. Это делается в ❶ в функции `loraSend()`.

Сейчас у нас есть два примера проектов, показывающих, как можно не только дистанционно управлять цифровыми выходами на гораздо большем, чем в ранних проектах, расстоянии, но и подтвердить выполнение действия. Теперь вы можете расширить эти примеры, чтобы создать свои проекты дистанционного управления. Далее мы поэкспериментируем с дистанционной передачей данных с датчиков через плату расширения LoRa.

Проект 49: беспроводная передача данных с датчиков с помощью LoRa

Этот проект основан на предыдущих, где мы использовали компьютер для получения замеров температуры от удаленного датчика.

Оборудование для передатчика

Для сборки передатчика нужно следующее оборудование:

- плата Arduino и кабель USB;
- плата расширения LoRa.

В этом проекте для управления используется монитор порта на вашем компьютере. Поэтому передатчик состоит только из палаты Arduino с платой расширения LoRa, подключенной к компьютеру через USB-кабель.

Оборудование для приемника

Для сборки приемника нужно следующее оборудование:

- плата Arduino и кабель USB;
- плата расширения LoRa;

- один датчик температуры TMP36;
- макетная плата для навесного монтажа;
- внешний источник питания для Arduino;
- провода со штекерами «штекер-гнездо».

Схема приемника

Наша схема состоит из единственного датчика температуры TMP36, подключенного к аналоговому входу A0, и платы расширения LoRa, которая вставлена контактами в разъемы на Arduino Uno (рис. 16.12).

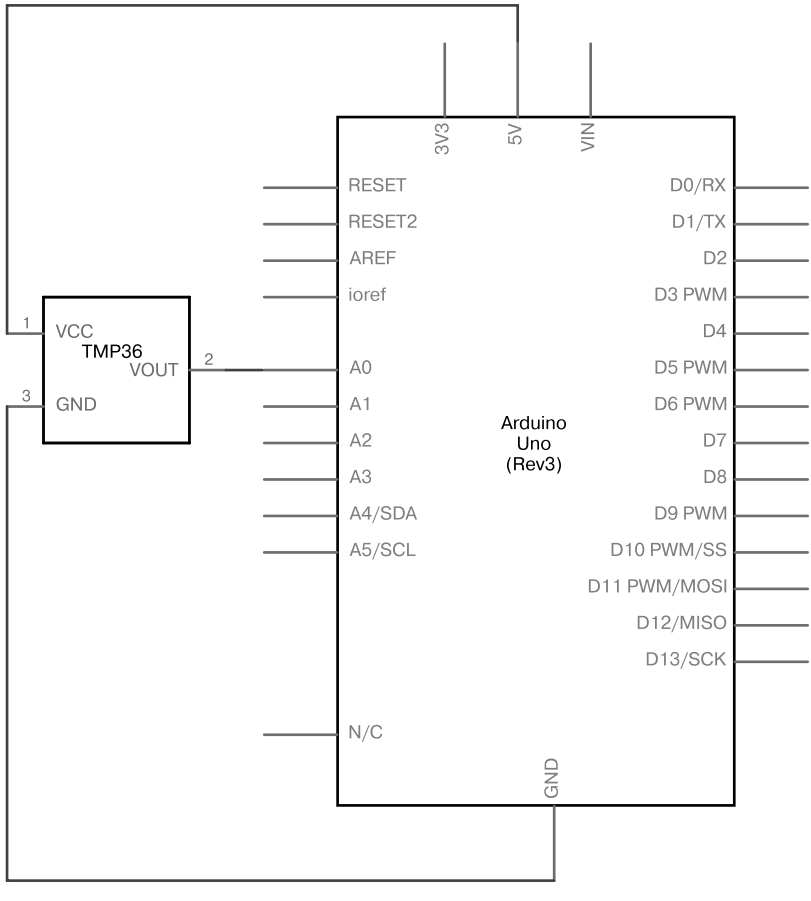


Рис. 16.12. Схема приемника для проекта 49

Приемник может быть на расстоянии от компьютера, поэтому для его питания можно использовать внешний источник или контейнер с батареями из предыдущих проектов.

Скетч передатчика

Теперь рассмотрим скетч передатчика. Введите и загрузите следующий скетч в плату Arduino, которая будет играть роль передатчика:

```
// Проект 49 — беспроводная передача данных с датчиков с помощью LoRa
// Скетч передатчика
#define LORAFREQ (915000000L)
#include <SPI.h>
#include <LoRa.h>

char command;

void loraSend(int controlCode)
{
  LoRa.beginPacket();      // Начать отправку данных
❶ LoRa.print("ABC");       // ABC — наш трехсимвольный код для передатчика
  LoRa.print(controlCode); // Послать управляющий код (в controlCode)
  LoRa.endPacket();        // Завершить отправку данных
  LoRa.receive();          // Начать прием
}

// Посылает в монитор порта текст, полученный от платы Arduino с датчиком
// температуры с помощью платы расширения LoRa
void takeAction(int packetSize)
{
  char incoming[31] = "";
  int k;
  for (int i = 0; i < packetSize; i++)
  {
    k = i;
    if (k > 31)
    {
      k = 31; // Гарантировать невозможность записи за пределы строки
    }
    incoming[k] = (char)LoRa.read();
    Serial.print(incoming[k]); // Вывести температуру, полученную от датчика
  }
  Serial.println();
}

void setup()
{
❷ LoRa.begin(LORAFREQ);       // Настроить частоту в LoRa
  LoRa.onReceive(takeAction); // Вызывать takeAction при получении данных
```

```
  LoRa.receive();             // Начать прием
  Serial.begin(9600);
}

void loop()
{
❸ Serial.print("Enter 1 for Celsius or 2 for Fahrenheit then Enter: ");
  Serial.flush(); // Удалить возможный "мусор" из буфера последовательного порта
❹ while (Serial.available() == 0)
  {
    // Ждать, пока что-то появится в буфере последовательного порта
  }
  while (Serial.available() > 0)
  {
    command = Serial.read() - '0';
    // Прочитать цифру из буфера,
    // вычесть из ASCII-кода цифры код символа '0', чтобы получить целое число
  }
  Serial.println();
❺ loraSend(command);
  delay(2000);
}
```

Как и в прошлых проектах из этой главы, скетч инициализирует оборудование LoRa и монитор порта ❷. Для приема команд пользователя и отправки их приемнику здесь вместо кнопок используется монитор порта. Скетч предлагает пользователю ввести `1` или `2` в поле ввода монитора порта, чтобы получить температуру в градусах Цельсия или Фаренгейта ❸. Компьютер ждет ввода пользователя ❹, а после отправляет приемнику введенную команду вызовом `loraSend()` ❺. И снова для идентификации приемника используется трехзначный код ❶.

Скетч приемника

Теперь перейдем к скетчу приемника. Введите и загрузите следующий скетч в плату Arduino, которая будет играть роль приемника:

```
// Проект 49 — беспроводная передача данных с датчиков с помощью LoRa
// Скетч приемника
#define LORAFREQ (915000000L)
#include <SPI.h>
#include <LoRa.h>

float sensor = 0;
float voltage = 0;
float celsius = 0;
float fahrenheit = 0;
```

```
// Посылает температуру в градусах Цельсия
void loraSendC()
{
  LoRa.beginPacket(); // Начать отправку данных
  sensor = analogRead(0);
  voltage = ((sensor * 5000) / 1024);
  voltage = voltage - 500;
  celsius = voltage / 10;
  fahrenheit = ((celsius * 1.8) + 32);
❶ LoRa.print("Temperature: ");
  LoRa.print(celsius, 2);
  LoRa.print(" degrees C");
❷ LoRa.endPacket();  // Завершить отправку данных
  LoRa.receive();    // Начать прием
}

// Посылает температуру в градусах Фаренгейта
void loraSendF()
{
  LoRa.beginPacket(); // Начать отправку данных
  sensor = analogRead(0);
  voltage = ((sensor * 5000) / 1024);
  voltage = voltage - 500;
  celsius = voltage / 10;
  fahrenheit = ((celsius * 1.8) + 32);
❶ LoRa.print("Temperature: ");
  LoRa.print(fahrenheit, 2);
  LoRa.print(" degrees F");
❷ LoRa.endPacket(); // Завершить отправку данных
  LoRa.receive();   // Начать прием
}

// Определяет, как обрабатываются данные, принятые платой расширения LoRa
void takeAction(int packetSize)
{
  char incoming[6] = "";
  int k;
  for (int i = 0; i < packetSize; i++)
  {
    k = i;
    if (k > 6)
    {
      k = 6; // Гарантировать невозможность записи за пределы строки
    }
    incoming[k] = (char)LoRa.read();
  }
❸ // Проверить трехзначный код, отправленный передатчиком
  if (incoming[0] != 'A')
```

```
  {
    return; // Если не 'A', прервать обработку и вернуться в void loop()
  }
  if (incoming[1] != 'B')
  {
    return; // Если не 'B', прервать обработку и вернуться в void loop()
  }
  if (incoming[2] != 'C')
  {
    return; // Если не 'C', прервать обработку и вернуться в void loop()
  }

  // Добравшись до этой точки, мы можем быть уверены в верности
  // трехзначного кода, полученного от передатчика
  if (incoming[3] == '1')
  {
❹   loraSendC();
  }
  if (incoming[3] == '2')
  {
❺   loraSendF();
  }
}

void setup()
{
  LoRa.begin(LORAFREQ);       // Настроить частоту в LoRa
  LoRa.onReceive(takeAction); // Вызывать takeAction при получении данных
  LoRa.receive();             // Начать прием
}

void loop()
{
}
```

Точно так же, как и в проекте 48, приемник проверяет трехзначный код, полученный от передатчика, чтобы убедиться, что данные предназначены для него, а затем анализирует управляющий код ❸. Если все полученные коды верны, приемник вызывает `loraSendC()` ❹ или `loraSendF()` ❺ для отправки температуры в градусах Цельсия или Фаренгейта. Эти две функции вычисляют температуру по значению, полученному от датчика TMP36. В строках ❶, ❷ они отправляют обратно передатчику текстовую строку с величиной и типом температуры.

После сборки обоих устройств и загрузки скетчей поместите приемник (с датчиком температуры) вместе с блоком питания туда, где вы хотите измерять температуру. Убедитесь, что передатчик подключен к компьютеру. Откройте Serial Monitor (Монитор порта) в IDE и следуйте инструкциям для получения температуры. Пример вывода можно увидеть на рис. 16.13.

```
Enter 1 for Celsius or 2 for Fahrenheit then Enter:
Temperature: 19.34 degrees C
Enter 1 for Celsius or 2 for Fahrenheit then Enter:
Temperature: 65.93 degrees F
Enter 1 for Celsius or 2 for Fahrenheit then Enter:
Temperature: 18.85 degrees C
Enter 1 for Celsius or 2 for Fahrenheit then Enter:
Temperature: 19.34 degrees C
Enter 1 for Celsius or 2 for Fahrenheit then Enter:
Temperature: 66.80 degrees F
Enter 1 for Celsius or 2 for Fahrenheit then Enter:
```

Рис. 16.13. Результаты проекта 49

Что дальше?

Эта глава показала, как просто реализовать дистанционное управление системами на основе Arduino. Можно управлять цифровыми выходами, посылая символы из одной платы Arduino в другую, и использовать плату расширения LoRa для создания более сложных систем управления несколькими платами Arduino, в том числе с получением ответов. Вооруженные полученными знаниями, вы можете смело творить.

Но на этом обсуждение приемов беспроводной передачи данных не заканчивается. В следующей главе вы узнаете, как можно организовать управление платой Arduino с помощью простого пульта от телевизора.

17

Инфракрасный пульт дистанционного управления

В этой главе вы:

- создадите и опробуете простой инфракрасный приемник;
- реализуете дистанционное управление цифровыми выходами Arduino;
- добавите дистанционное управление в модель робота из главы 14.

Здесь вы увидите, как с помощью недорогого модуля приемника плата Arduino принимает сигналы от инфракрасного пульта дистанционного управления и реагирует на них.

Что такое инфракрасный пульт дистанционного управления

Многие из нас каждый день пользуются инфракрасными пультами дистанционного управления. Но не каждый знает принцип их работы. Инфракрасные (ИК) сигналы — это пучки света инфракрасного диапазона. Они не воспринимаются невооруженным глазом. Поэтому, если посмотреть на маленький светодиод на пульте дистанционного управления и нажать какую-нибудь кнопку, вы не увидите свечения.

В ИК-пультах дистанционного управления есть один или несколько специальных светодиодов, генерирующих инфракрасный свет, которые передают ИК-сигналы. Если нажать кнопку на пульте управления, светодиод многократно включается и выключается, воспроизводя определенные последовательности, уникальные для каждой кнопки. Этот сигнал принимается специальным ИК-приемником и преобразуется в электрические импульсы. Они, в свою очередь, преобразуются электроникой

приемника в данные. Если вам интересно увидеть световые импульсы, посмотрите на ИК-светодиод с помощью камеры смартфона или цифровой камеры.

Подготовка к приему ИК-сигналов

Для начала установим библиотеку IRremote для Arduino, поддерживающую работу с ИК-приемниками. Для этого воспользуйтесь ссылкой https://github.com/z3t0/Arduino-IRremote/archive/master.zip, чтобы загрузить архив. Затем установите его, как описывается в главе 7.

ИК-приемник

Следующий шаг — установка ИК-приемника и проверка его работоспособности. Вы можете выбрать отдельный ИК-приемник (рис. 17.1) или готовый модуль с разъемом (рис. 17.2), который проще в использовании.

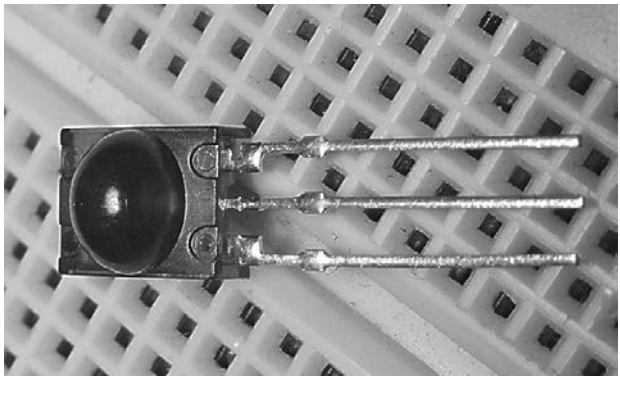


Рис. 17.1. ИК-приемник

Рис. 17.2. Готовый модуль с разъемом

На рис. 17.1 показан отдельный ИК-приемник Vishay TSOP4138. Нижний вывод соединяется с цифровым контактом на плате Arduino, центральный — с контактом GND и верхний — с контактом 5 V.

На рис. 17.2 показан готовый модуль с разъемом. Его можно приобрести у PMD Way или других продавцов. Преимущество готовых модулей — в наличии разъема и маркировки, упрощающих подключение.

Независимо от модели, которую вы выберете для себя, во всех следующих проектах вывод D (или data line — «линия данных») приемника будет подключаться к цифровому контакту 2 на плате Arduino, вывод VCC — к 5 V и вывод GND — к GND.

Пульт дистанционного управления

Теперь вам нужен пульт дистанционного управления. Я использовал пульт от телевизора Sony (рис. 17.3). Если у вас нет такого, можете использовать любой недорогой универсальный пульт, поддерживающий коды управления Sony. Как его настроить, должно быть описано в инструкции к пульту.

Рис. 17.3. Типичный пульт дистанционного управления Sony

Тестовый скетч

Теперь проверим работоспособность всего комплекта оборудования. После подключения ИК-приемника к плате Arduino введите и загрузите скетч из листинга 17.1.

Листинг 17.1. Тест ИК-приемника

```
❶ #include <IRremote.h>       // Подключить библиотеку

❷ IRrecv irrecv(receiverpin); // Создать экземпляр объекта IRrecv
❸ decode_results results;
```

```
int receiverpin = 2;  // Вывод 1 ИК-приемника — к цифровому
                      // входу 2 на плате Arduino

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Запустить ИК-приемник
}

void loop()
{
❹ if (irrecv.decode(&results))        // ИК-сигнал принят?
  {
❺   Serial.print(results.value, HEX); // Вывести ИК-код на монитор порта
    Serial.print(" ");
    irrecv.resume();              // Разрешить прием следующего значения
  }
}
```

Этот скетч несложен, потому что основная работа выполняется библиотекой IRremote. В строке ❹ проверяется, был ли получен сигнал от пульта. Если да, то код сигнала выводится в шестнадцатеричном виде в монитор порта ❺. В строках ❶, ❷ и ❸ активируется библиотека IRremote и создается экземпляр библиотечного объекта для использования в скетче.

Тестирование собранного устройства

После загрузки скетча откройте монитор порта, наведите пульт управления на приемник и начинайте нажимать кнопки. При нажатии каждой кнопки вы должны увидеть коды в окне монитора порта. На рис. 17.4 показаны результаты однократного нажатия кнопок 1, 2 и 3.

Рис. 17.4. Результаты нажатия кнопок после запуска скетча из листинга 17.1

В табл. 17.1 перечислены коды, генерируемые простым дистанционным пультом управления Sony, которые будут использоваться в других скетчах. Проверяя скетч в листинге 17.1, обратите внимание, что каждый код появляется трижды. Это особенность ИК-систем Sony, которые посылают код три раза для каждого нажатия кнопки. Вы можете игнорировать эти повторы с помощью умного кода. А сейчас перейдем к реализации следующего проекта с дистанционным управлением.

Таблица 17.1. Примеры ИК-кодов Sony

Кнопка	Код	Кнопка	Код
Power (Вкл./выкл.)	A90	7	610
Mute (Выкл. звук)	290	8	У10
1	10	9	110
2	810	0	910
3	410	Volume up (Увеличить громкость)	490
4	C10	Volume down (Уменьшить громкость)	C90
5	210	Channel up (Следующий канал)	90
6	A10	Channel down (Предыдущий канал)	890

Проект 50: дистанционное управление Arduino с помощью ИК-пульта

Этот проект покажет, как управлять цифровыми выходами, используя ИК-пульт. Здесь мы реализуем управление цифровыми контактами с 3 по 7 с помощью цифровых кнопок с 3 по 7 на пульте Sony. После нажатия кнопки на соответствующем цифровом выходе пульта устанавливается уровень `HIGH` на одну секунду, а потом возвращается уровень `LOW`. Вы можете использовать этот проект как основу или руководство для добавления поддержки удаленного управления в других проектах.

Оборудование

Для этого проекта нам нужны:

- плата Arduino и кабель USB;
- пять светодиодов;
- пять резисторов номиналом 560 Ом;
- ИК-приемник или модуль;
- одна макетная плата;
- несколько отрезков провода разной длины.

Схема

Схема состоит из ИК-приемника с выходом, подключенным к цифровому контакту 2, и пяти светодиодов с ограничивающими резисторами, подключенными к цифровым контактам с 3 по 7 включительно (рис. 17.5).

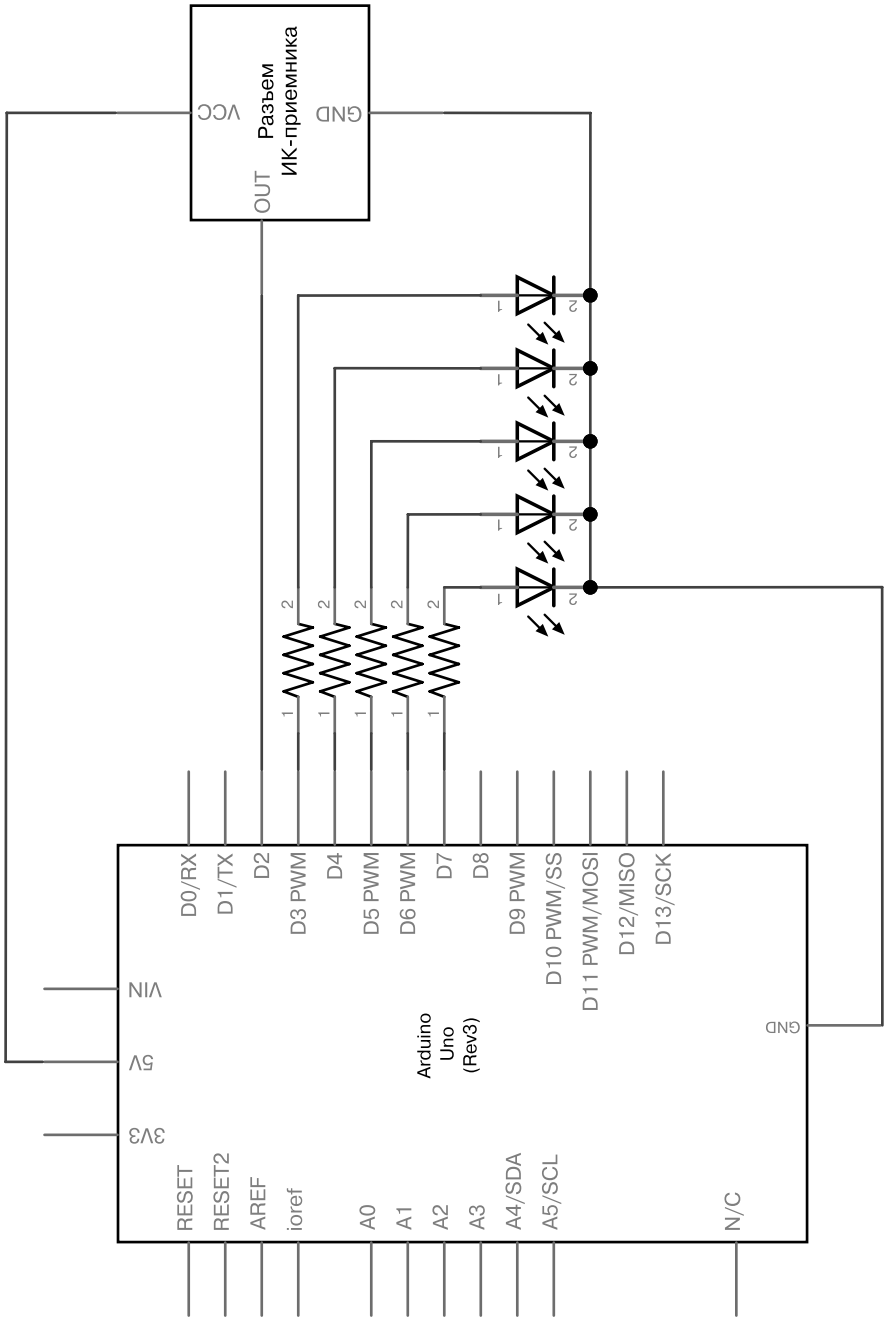


Рис. 17.5. Принципиальная схема для проекта 50

Скетч

Введите и загрузите следующий скетч:

```
// Проект 50 – дистанционное управление Arduino с помощью ИК-пульта

#include <IRremote.h>
IRrecv irrecv(receiverpin); // Создать экземпляр объекта IRrecv
decode_results results;
int receiverpin = 2;        // Вывод 1 ИК-приемника – к цифровому
                            // входу 2 на плате Arduino

void setup()
{
  irrecv.enableIRIn();          // Запустить ИК-приемник
  for (int z = 3 ; z < 8 ; z++) // Настроить цифровые выходы
  {
    pinMode(z, OUTPUT);
  }
}

❶ void translateIR()
  // Принимает ИК-коды Sony
  // и выполняет соответствующие им операции
  {
    switch(results.value)
    {
❷     case 0x410: pinOn(3); break; // 3
      case 0xC10: pinOn(4); break; // 4
      case 0x210: pinOn(5); break; // 5
      case 0xA10: pinOn(6); break; // 6
      case 0x610: pinOn(7); break; // 7
    }
  }

❸ void pinOn(int pin) // Включает контакт pin на одну секунду
  {
    digitalWrite(pin, HIGH);
    delay(1000);
    digitalWrite(pin, LOW);
  }

  void loop()
  {
❹   if (irrecv.decode(&results))    // ИК-сигнал принят?
    {
      translateIR();
❺     for (int z = 0 ; z < 2 ; z++) // Игнорировать второе и третье повторение
      {
        irrecv.resume();            // Разрешить прием следующего значения
      }
    }
  }
```

Этот скетч делится на три основные части. Первая ожидает сигнал от пульта управления ❹. После приема сигнала полученный код передается в функцию `translateIR()` ❶ для определения нажатой кнопки и выполнения соответствующей операции.

Заметьте, как в инструкции `switch-case` ❷ сравниваются шестнадцатеричные коды, возвращаемые библиотекой IRremote. Это те самые коды, которые вернул тестовый скетч из листинга 17.1. Если принят код, соответствующий одной из кнопок с 3 по 7, вызывается функция `pinOn()` ❸. Она устанавливает высокий уровень на соответствующем цифровом выходе на одну секунду.

Мы уже знаем, что в ответ на однократное нажатие кнопки пульты Sony посылают код трижды. Поэтому в скетч добавлен короткий цикл ❺, позволяющий пропустить второй и третий коды. Обратите внимание на дополнительную приставку `0x` в начале шестнадцатеричных чисел в инструкциях `case` ❷.

ПРИМЕЧАНИЕ

Шестнадцатеричные числа — это числа в системе счисления с основанием 16. Для их записи используются цифры от 0 до 9 и буквы от A до F. Например, десятичное число 10 в шестнадцатеричном виде записывается как A, десятичное число 15 — как F, 16 — как 10 и т. д. Если в скетчах используются шестнадцатеричные числа, они должны начинаться с 0x.

Расширение возможностей

Добавив в скетч анализ дополнительных кнопок, можно организовать управление движением нашей модели робота. Для этого достаточно с помощью скетча из листинга 17.1 определить коды соответствующих кнопок на пульте и добавить реакцию на новые коды в инструкцию `switch-case`.

Проект 51: дистанционное ИК-управление моделью робота

Чтобы показать, как внедрить дистанционное управление с помощью ИК-пульта в проект, добавим эту возможность в модель робота. Ее создание описано в проекте 39 (см. главу 14). Дальше вы узнаете, как организовать управление движением робота с помощью простого телевизионного пульта Sony.

Оборудование

Чтобы показать вам, как внедрить ИК-пульт дистанционного управления в проект, мы добавим пульт к роботу из проекта 39 в главе 14. Следующий скетч реализует реакцию робота на нажатия кнопок на пульте управления: кнопка 2 — вперед, 8 — назад, 4 — поворот влево и 6 — поворот вправо.

Скетч

После подключения ИК-приемника к модели робота введите и загрузите следующий скетч:

```
// Проект 51 – дистанционное ИК-управление моделью робота
#include <IRremote.h>
int receiverpin = 2; // Вывод 1 ИК-приемника – к цифровому
                     // входу 2 на плате Arduino
IRrecv irrecv(receiverpin); // Создать экземпляр объекта IRrecv
decode_results results;

#include <AFMotor.h>
AF_DCMotor motor1(1); // Настройка экземпляров, представляющих моторы
AF_DCMotor motor2(2);
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);

void goForward(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void goBackward(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}
```

```
void rotateLeft(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(FORWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(FORWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void rotateRight(int speed, int duration)
{
  motor1.setSpeed(speed);
  motor2.setSpeed(speed);
  motor3.setSpeed(speed);
  motor4.setSpeed(speed);
  motor1.run(BACKWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(BACKWARD);
  delay(duration);
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

// translateIR выполняет операцию, соответствующую ИК-коду,
// используются коды Sony
void translateIR()
{
  switch (results.value)
  {
    case 0x810:
      goForward(255, 250);
      break; // 2
    case 0xC10:
      rotateLeft(255, 250);
      break; // 4
    case 0xA10:
      rotateRight(255, 250);
      break; // 6
    case 0xE10:
      goBackward(255, 250);
```

```
      break; // 8
  }
}

void setup()
{
  delay(5000);
  irrecv.enableIRIn(); // Запустить ИК-приемник
}

void loop()
{
  if (irrecv.decode(&results))    // ИК-сигнал принят?
  {
    translateIR();
    for (int z = 0 ; z < 2 ; z++) // Игнорировать повторения
    {
      irrecv.resume();            // Разрешить прием следующего значения
    }
  }
}
```

Этот скетч уже знаком вам. Но вместо включения цифровых выходов он вызывает функции управления электродвигателями, которые использовались в скетчах управления роботом в главе 14.

Что дальше?

С помощью проектов из этой главы вы научились посылать команды плате Arduino с помощью ИК-пульта управления. Объединив все знания (в том числе и те, что вы получите в следующих проектах), вы сможете заменить физические формы ввода, например кнопки, удаленным управлением.

Но наши развлечения на этом не заканчиваются. В следующей главе мы будем использовать Arduino для реализации систем радиочастотной идентификации, которые могут казаться завораживающими и футуристическими.

18

Чтение радиомаркеров RFID

В этой главе вы:

- узнаете, как реализовать радиочастотную идентификацию с помощью Arduino;
- освоите сохранение данных в EEPROM Arduino;
- начнете создавать систему доступа по радиомаркерам на основе Arduino.

Радиочастотная идентификация (Radio-Frequency Identification, RFID) — это беспроводная система, использующая электромагнитное поле для передачи данных от одного объекта к другому без их прямого взаимодействия. Вы можете сконструировать устройство на основе Arduino для создания системы доступа и управления цифровыми выходами, которое будет читать радиомаркеры и карты RFID. Возможно, вам приходилось использовать карты RFID, например, радиоключи для отпирания дверей или карты оплаты услуг общественного транспорта, которые вставляются в считыватель при входе. На рис. 18.1 показаны примеры радиоключей и карт RFID.

Внутреннее устройство радиомаркеров

Радиомаркер RFID состоит из маленькой микросхемы с памятью, доступной для специального устройства чтения. У большинства радиомаркеров нет внутреннего источника питания. Они питаются энергией электромагнитного поля, генерируемого устройством чтения RFID. Это поле создается миниатюрной катушкой индуктивности, которая также играет роль антенны для передачи данных между радиомаркером и устройством чтения. На рис. 18.2 показана катушка-антенна устройства чтения радиомаркеров RFID, которое будет использоваться в этой главе.

Рис. 18.1. Примеры устройств RFID

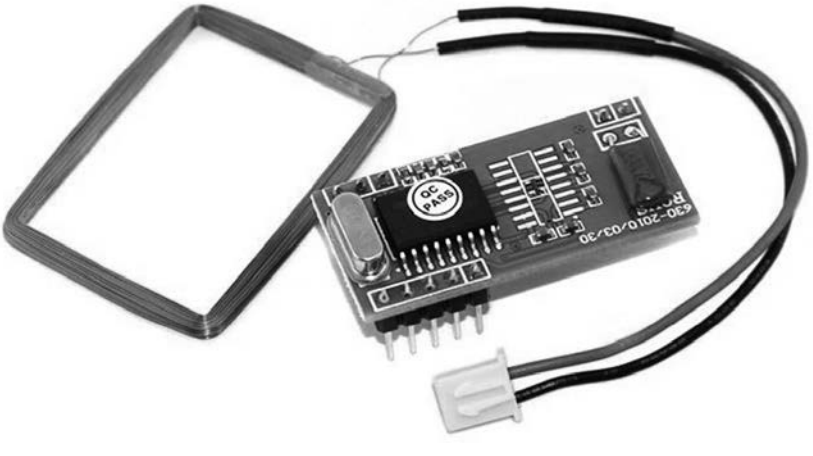


Рис. 18.2. Устройство чтения радиомаркеров RFID

Устройство чтения из этой главы (артикул 113990014) можно приобрести в PMD Way. Оно недорогое, простое в использовании и действует на частоте 125 кГц. Выбирая радиомаркер RFID, приобретайте модель, действующую на той же частоте. Их можно найти на https://pmdway.com/collections/rfid-tags/.

Проверка оборудования

В этом разделе мы подключим устройство чтения радиомаркеров RFID к плате Arduino и проверим его работоспособность с помощью простого скетча. Он будет читать данные с радиомаркера RFID и посылать их в монитор порта. Во избежание конфликта с последовательным портом, связывающим ПК и Arduino, устройство чтения RFID будет подключено к другим цифровым контактам. Для связи с ним будет использована библиотека SoftwareSerial, как мы уже делали это в главе 15 с модулем GPS-приемника.

Схема

На рис. 18.3 показано расположение контактов на модуле RFID (вид сверху).

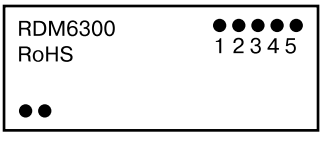


Рис. 18.3. Контакты на модуле RFID

Проверка

Чтобы установить соединение между считывателем RFID и Arduino, выполните следующие действия, используя перемычки типа «гнездо-штекер».

1. Подключите разъем катушки-антенны к контактам внизу слева на плате модуля чтения RFID. У них нет полярности, поэтому они могут подключаться как угодно.
2. Соедините контакт GND модуля RFID с контактом GND на плате Arduino.
3. Соедините контакт 1 (5 В) модуля RFID с контактом 5 V на плате Arduino.
4. Соедините контакт 4 (прием) модуля RFID с контактом D3 на плате Arduino.
5. Соедините контакт 5 (передача) модуля RFID с контактом D2 на плате Arduino.

Тестовый скетч

Введите и загрузите скетч из листинга 18.1.

Листинг 18.1. Скетч для проверки работоспособности модуля RFID

```
#include <SoftwareSerial.h>
SoftwareSerial Serial2(2, 3);
int data1 = 0;

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

```
  Serial2.begin(9600);
}

void loop()
{
  if (Serial2.available() > 0) {
    data1 = Serial2.read();
    // Вывести считанное число
    Serial.print(" ");
    Serial.print(data1, DEC);
  }
}
```

Исследование вывода в окне монитора порта

Откройте окно монитора порта и поднесите радиомаркер RFID к катушке. В мониторе появятся значения, похожие на изображенные на рис. 18.4.

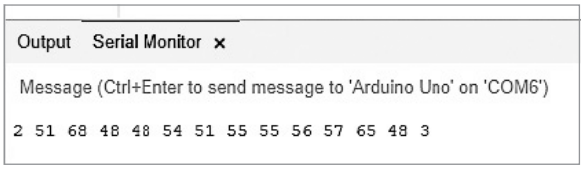


Рис. 18.4. Пример вывода скетча из листинга 18.1

Обратите внимание, что в окне монитора порта отображается 14 чисел. Это уникальный числовой идентификатор радиомаркера RFID, который будет использоваться в будущих скетчах для распознавания радиомаркера. Запишите числа для всех имеющихся у вас радиомаркеров. Это пригодится в следующих проектах.

Проект 52: простая RFID-система контроля доступа

Теперь попробуем применить систему RFID на практике. В этом проекте вы узнаете, как инициировать событие Arduino при считывании одной из двух правильных меток RFID. Наш скетч хранит идентификационные числа двух радиомаркеров RFID. Для обоих он будет выводить в монитор порта текст Accepted (Доступ разрешен). Если поднести к антенне любой другой радиомаркер, скетч выведет в монитор порта текст Rejected (Доступ запрещен). Этот скетч будет основой для добавления функций управления доступом по радиомаркерам RFID в существующие проекты.

Скетч

Введите и загрузите следующий скетч. Но во время ввода замените символы x в массивах ❶ и ❷ числами, соответствующими вашим радиомаркерам. Вы должны были их записать, как советовалось выше (мы обсуждали массивы в главе 6).

```
// Проект 52 — простая RFID-система контроля доступа
#include <SoftwareSerial.h>
SoftwareSerial Serial2(2, 3);
int data1 = 0;
int ok = -1;

// Чтобы определить числовые идентификаторы радиомаркеров,
// воспользуйтесь скетчем из листинга 18.1
❶ int tag1[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};
❷ int tag2[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};

// Следующий массив используется для чтения и сравнения
int newtag[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};

❸ boolean comparetag(int aa[14], int bb[14])
{
  boolean ff = false;
  int fg = 0;
  for (int cc = 0 ; cc < 14 ; cc++)
  {
    if (aa[cc] == bb[cc])
    {
      fg++;
    }
  }
  if (fg == 14)
  {
    ff = true;
  }
  return ff;
}

// Сравнивает известные числовые идентификаторы
// с только что прочитанным
❹ void checkmytags()
{
  ok = 0; // Эта переменная помогает принять решение
          // и может иметь три значения:
          // 1 — числовой идентификатор прочитан и совпадает
          // с одним из известных;
          // 0 — числовой идентификатор прочитан,
          // но не совпадает ни с одним из известных;
          // -1 — числовой идентификатор не был прочитан
```

```
    if (comparetag(newtag, tag1) == true)
    {
❺     ok++;
    }
    if (comparetag(newtag, tag2) == true)
    {
❻     ok++;
    }
  }

  void setup()
  {
    Serial.begin(9600);
    Serial2.begin(9600);
    Serial2.flush(); // Очистить буфер последовательного порта,
                     // иначе первая попытка чтения может дать
                     // ошибочный результат
  }

  void loop()
  {
    ok = -1;
    if (Serial2.available() > 0) // Если была попытка чтения
    {
      // Прочитать число из модуля RFID
      delay(100); // Дать время поступить всем данным
                  // в буфер последовательного порта
❼     for (int z = 0 ; z < 14 ; z++) // Прочитать числовой
                                     // идентификатор радиомаркера
      {
        data1 = Serial2.read();
        newtag[z] = data1;
      }
      Serial2.flush(); // Аннулировать повторные попытки чтения
      // Теперь сравнить числовые идентификаторы
❽     checkmytags();
    }
❾   // Выполнить необходимые операции по результатам
    if (ok > 0) // Прочитан известный радиомаркер
    {
      Serial.println("Accepted");
      ok = -1;
    }
    else if (ok == 0) // Если прочитан неизвестный радиомаркер
    {
      Serial.println("Rejected");
      ok = -1;
    }
  }
```

Принцип действия

Если поднести радиомаркер RFID к антенне, устройство чтения перешлет числовой идентификатор через последовательный порт. Скетч принимает все 14 чисел и помещает их в массив `newtag[]` ❼. Потом прочитанный идентификатор сравнивается с двумя известными, ❶ и ❷, в функции `checkmytags()` (❹ и ❽). При этом фактическое сравнение массивов выполняется в функции `comparetag()` ❸.

Функция `comparetag()` принимает два числовых массива и возвращает логический признак их совпадения (`true`) или несовпадения (`false`). При обнаружении совпадения переменной `ok` присваивается значение `1` (❺ и ❻). В ❾ в зависимости от результата совпадения прочитанного идентификатора с одним из известных выполняются определенные операции.

После загрузки скетча откройте окно монитора последовательного порта и подносите к антенне разные радиомаркеры. Вы должны получить результаты как на рис. 18.5.

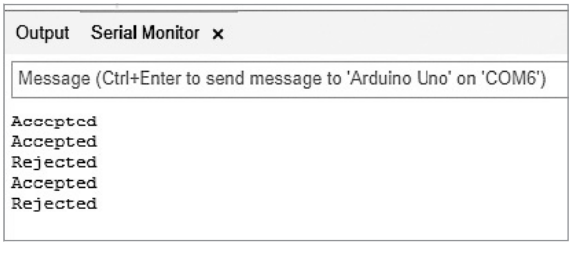


Рис. 18.5. Результаты работы проекта 52

Сохранение данных во встроенном EEPROM

Когда вы определяете и используете переменную в своих эскизах Arduino, данные сохраняются только до сброса Arduino или отключения питания. Но как быть, если нужно сохранить значение для использования в будущем? Например, изменяемый пользователем секретный код для кодового замка из главы 11. Для этой цели используется EEPROM (electrically erasable read-only memory — электрически стираемое программируемое постоянное запоминающее устройство). EEPROM находится внутри микроконтроллера ATmega328 и не теряет хранящиеся в нем данные при выключении питания.

EEPROM в Arduino может хранить до 1024 байт данных в ячейках от 0 до 1023. Напомню, что один байт хранит целое число в диапазоне от 0 до 255. Позднее вы узнаете, почему такая организация прекрасно подходит для хранения числовых идентификаторов радиомаркеров RFID. Чтобы использовать EEPROM, нужно

подключить библиотеку EEPROM (входит в состав Arduino IDE) в скетче, как показано ниже:

```
#include <EEPROM.h>
```

Выполнить запись значения в EEPROM:

```
EEPROM.write(a, b);
```

Здесь параметр *a* определяет номер ячейки (от 0 до 1023) для записи значения, а параметр *b* — байт данных для записи в ячейку с номером *a*.

Для извлечения данных из EEPROM используется следующая функция:

```
value = EEPROM.read(position);
```

Эта инструкция извлечет байт данных из ячейки EEPROM с номером *position* и запишет его в переменную *value*.

ПРИМЕЧАНИЕ

У EEPROM ограниченный срок использования, поэтому в итоге хранящиеся в нем данные могут быть утеряны! Согласно утверждениям производителя Atmel, EEPROM может выдержать до 100 000 циклов стирания/записи в каждую ячейку. Число операций чтения не ограничивается.

Чтение и запись в EEPROM

Ниже приводится пример чтения и записи данных в EEPROM. Введите и загрузите скетч из листинга 18.2.

Листинг 18.2. Скетч, демонстрирующий использование EEPROM

```
#include <EEPROM.h>

int zz;

void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop()
{
  Serial.println("Writing random numbers...");
  for (int i = 0; i < 1024; i++)
  {
    zz = random(255);
```

```
❶     EEPROM.write(i, zz);
    }
    Serial.println();
    for (int a = 0; a < 1024; a++)
    {
❷     zz = EEPROM.read(a);
      Serial.print("EEPROM position: ");
      Serial.print(a);
      Serial.print(" contains ");
❸     Serial.println(zz);
      delay(25);
    }
  }
```

В цикле ❶ в каждую ячейку EEPROM записывается случайное число от 0 до 255. Во втором цикле ❷ сохраненные значения извлекаются и отображаются в окне монитора порта ❸.

После загрузки скетча откройте окно монитора порта. Вы должны увидеть то, что изображено на рис. 18.6.

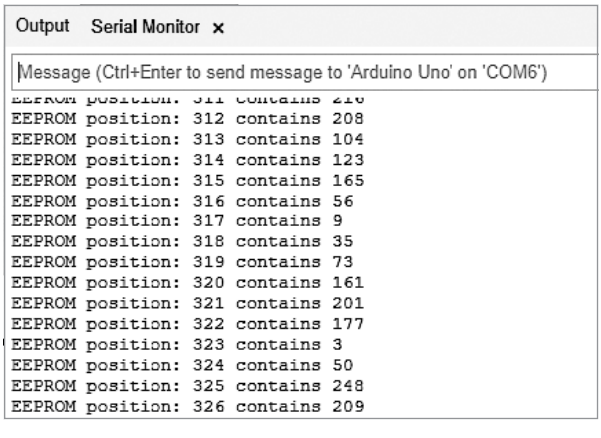


Рис. 18.6. Пример вывода скетча из листинга 18.2

Теперь вы готовы к обсуждению проекта с использованием EEPROM.

Проект 53: RFID-система управления с запоминанием последнего действия

Из проекта 52 мы узнали, как можно использовать RFID для управления освещением или электрическим замком. Но тогда предполагалось, что система не должна ничего запоминать при выключении питания или сбросе. К примеру, если после

включения освещения устройство на основе Arduino будет выключено, то выключится и освещение. Но зачастую нужно, чтобы после повторного включения плата Arduino «вспоминала» состояние, которое было к моменту отключения питания, и восстанавливала его. Давайте решим эту задачу.

В нашем проекте последнее действие будет запоминаться в EEPROM (например, «закрыто» или «открыто»). Когда скетч запустится после восстановления питания или сброса платы Arduino, система восстановит предыдущее состояние, хранящееся в EEPROM.

Скетч

Введите и загрузите следующий скетч. Снова замените символы x в массивах ❶ и ❷ числами, соответствующими вашим радиомаркерам, как в проекте 52.

```
// Проект 53 — RFID-система управления с запоминанием
//             последнего действия

#include <SoftwareSerial.h>
SoftwareSerial Serial2(2, 3);
#include <EEPROM.h>

int data1 = 0;
int ok = -1;
int lockStatus = 0;

// Чтобы определить числовые идентификаторы радиомаркеров,
// воспользуйтесь скетчем из листинга 18.1
❶ int tag1[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};
❷ int tag2[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};

// Следующий массив используется для чтения и сравнения
int newtag[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};

// comparetag сравнивает два массива и возвращает true,
// если они идентичны; она отлично подходит
// для сравнения числовых идентификаторов RFID
boolean comparetag(int aa[14], int bb[14])
{
  boolean ff = false;
  int fg = 0;

  for (int cc = 0; cc < 14; cc++)
  {
    if (aa[cc] == bb[cc])
    {
      fg++;
    }
  }
```

```
  if (fg == 14)
  {
    ff = true;
  }
  return ff;
}

// Сравнивает известные числовые идентификаторы
// с только что прочитанным
void checkmytags()
{
  ok = 0;
  if (comparetag(newtag, tag1) == true)
  {
    ok++;
  }
  if (comparetag(newtag, tag2) == true)
  {
    ok++;
  }
}

❸ void checkLock()
{
  Serial.print("System Status after restart ");
  lockStatus = EEPROM.read(0);
  if (lockStatus == 1)
  {
    Serial.println("- locked");
    digitalWrite(13, HIGH);
  }
  if (lockStatus == 0)
  {
    Serial.println("- unlocked");
    digitalWrite(13, LOW);
  }
  if ((lockStatus != 1) && (lockStatus != 0))
  {
    Serial.println("EEPROM fault - Replace Arduino hardware");
  }
}

void setup()
{
  Serial.begin(9600);
  Serial2.begin(9600);
  Serial2.flush(); // Необходимо для очистки буфера
  pinMode(13, OUTPUT);
❹  checkLock();
}
```

```
void loop()
{
  ok = -1;
  if (Serial2.available() > 0) // Если была попытка чтения
  {
    // прочитать число из модуля RFID
    delay(100);
    for (int z = 0 ; z < 14 ; z++) // Прочитать числовой
                                   // идентификатор радиомаркера
    {
      data1 = Serial2.read();
      newtag[z] = data1;
    }
    Serial2.flush(); // Аннулировать повторные попытки чтения
    // Теперь сравнить числовые идентификаторы
    checkmytags();
  }
❺ if (ok > 0) // Прочитан известный радиомаркер
  {
    lockStatus = EEPROM.read(0);
    if (lockStatus == 1) // Если заперт — отпереть
    {
❻     Serial.println("Status - unlocked");
      digitalWrite(13, LOW);
      EEPROM.write(0, 0);
    }
    if (lockStatus == 0)
❼   {
      Serial.println("Status - locked");
      digitalWrite(13, HIGH);
      EEPROM.write(0, 1);
    }
    if ((lockStatus != 1) && (lockStatus != 0))
    {
❽     Serial.println("EEPROM fault - Replace Arduino hardware");
    }
  }
  else if (ok == 0) // Если прочитан неизвестный радиомаркер
  {
    Serial.println("Incorrect tag");
    ok = -1;
  }
  delay(500);
}
```

Принцип действия

Этот скетч — измененная версия скетча из проекта 52. Здесь с помощью встроенного светодиода имитируется состояние замка, который запирается (включается) или отпирается (выключается) при поднесении знакомого радиомаркера RFID к антенне модуля чтения.

После чтения и сопоставления числового идентификатора RFID ❺ состояние замка изменяется. Состояние сохраняется в первой ячейке EEPROM и может иметь два значения: `0` — «открыт» и `1` — «закрыт». Каждый раз, когда скетч получает идентификатор знакомого радиомаркера, он изменяет состояние замка на противоположное («открыт» на «закрыт» и обратно на «открыт») в ❻ и ❼.

В скетч была добавлена обработка ошибки на случай, если значение в EEPROM стерлось. Когда извлеченное из EEPROM значение не равно 0 или 1, в монитор порта выводится предупреждающее сообщение ❽. В момент повторного запуска скетча после сброса вызывается функция `checkLock()` (❶, ❷, ❸ и ❹). Она читает значение из ячейки в EEPROM, определяет последнее хранящееся состояние и восстанавливает состояние замка («закрыт» или «открыт»).

Что дальше?

И снова мы увидели, как просто с Arduino реализуются сложные проекты. Теперь вы сможете добавлять в свои проекты поддержку технологии RFID и создавать системы контроля доступа профессионального уровня, управляя цифровыми выходами легким взмахом карты RFID. Мы еще раз рассмотрим все это в главе 20.

19

Шины данных

В этой главе вы:

- познакомитесь с шиной I²C;
- узнаете, как использовать EEPROM (ЭСППЗУ — электрически стираемое программируемое постоянное запоминающее устройство) и расширитель порта на шине I²C;
- познакомитесь с шиной SPI;
- научитесь использовать цифровой реостат на шине SPI.

Arduino взаимодействует с другими устройствами при помощи *шины данных*, системы соединений, позволяющей двум и более устройствам упорядоченно обмениваться данными. Шина данных используется для подключения к плате Arduino разных датчиков, расширительных устройств ввода/вывода и других компонентов.

Наиболее значимы для Arduino две шины: *шина последовательного периферийного интерфейса* (Serial Peripheral Interface, SPI) и *шина связи интегральных схем* (Inter-Integrated Circuit, I²C). Многие датчики и внешние устройства поддерживают обмен данных по ним.

Шина I²C

Шина I²C, также известная как *двухпроводной интерфейс* (Two Wire Interface, TWI) — простая и удобная шина для обмена данными. Данные между устройствами и Arduino передаются по двум линиям, которые называют *линией данных* (Serial Data Line, SDA) и *тактовой линией* (Serial Clock Line, SCL). В модели Arduino Uno линия SDA выведена на контакт A4, а линия SCL — на контакт A5 (рис. 19.1).

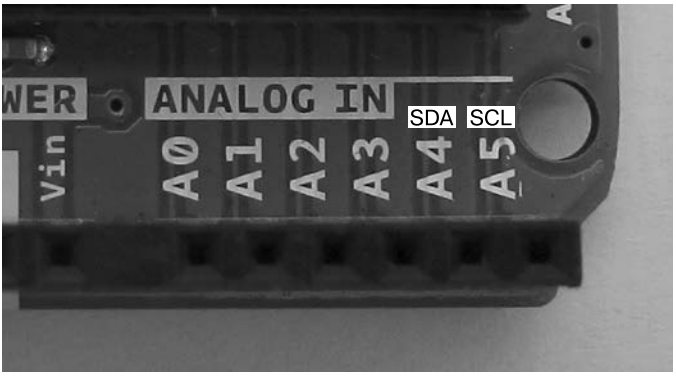


Рис. 19.1. Контакты на плате Arduino Uno, служащие одновременно выводами шины I²C

В некоторых новейших платах R3 есть отдельные контакты для подключения к шине I²C. Они находятся в верхнем левом углу для удобства доступа к ним, как показано на рис. 19.2. Если на вашей плате есть эти контакты, вы сможете использовать контакты A4 и A5 для любых других целей.

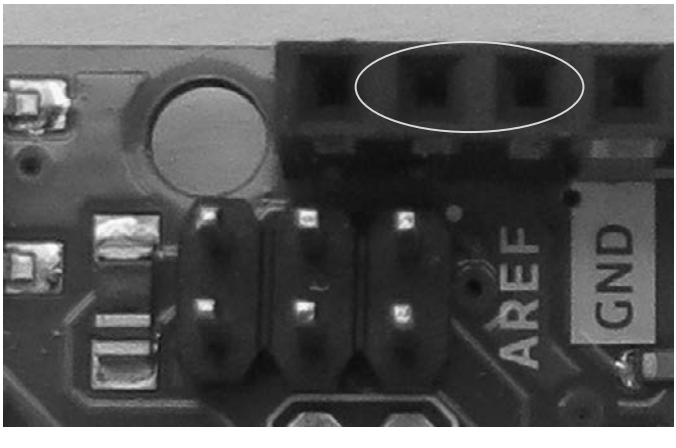


Рис. 19.2. Отдельные контакты для подключения к шине I²C

Место, обычно используемое для размещения маркировки, занимают шесть контактов. Они предназначены для перепрограммирования дополнительного микроконтроллера, с помощью которого мы программируем основной микроконтроллер Arduino по интерфейсу USB. Поэтому подписи отдельных контактов шины I²C нанесены на обратной стороне платы (рис. 19.3).

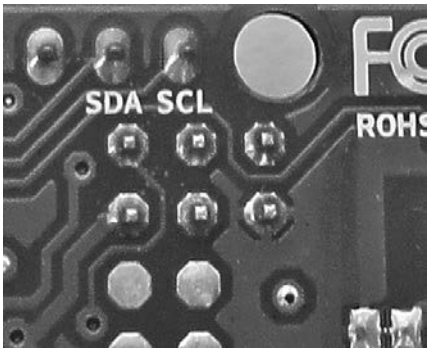


Рис. 19.3. Подписи отдельных контактов шины I²C

Будучи подключенной к шине I²C, плата Arduino считается *ведущим устройством*, а все остальные — *ведомыми*. Каждое ведомое устройство имеет свой адрес, выраженный шестнадцатеричным числом. Он позволяет плате Arduino обращаться и взаимодействовать с каждым устройством по отдельности. Обычно у устройства есть 7-битный адрес I²C, который указан в документации к нему. Конкретные доступные адреса определяются подключением контактов IC определенным образом.

ПРИМЕЧАНИЕ

Arduino питается напряжением 5 В, поэтому подключаемые к ней устройства I²C тоже должны питаться напряжением 5 В или быть совместимы с таким напряжением. Обязательно уточните эту информацию перед покупкой.

Чтобы использовать шину I²C, сначала подключите библиотеку Wire (входит в состав Arduino IDE):

```
#include <Wire.h>
```

В функции `void setup()` активируйте шину:

```
Wire.begin();
```

Данные по шине передаются по одному байту за раз. Чтобы послать байт из платы Arduino в устройство на шине, нужно вызвать три функции.

1. Первая инициализирует связь, как показано ниже (где аргумент *`address`* — это адрес ведомого устройства на шине в шестнадцатеричном виде, например `0x50`):

   ```
   Wire.beginTransmission(address);
   ```

2. Вторая посылает 1 байт данных из Arduino в устройство с адресом, указанным в предыдущем вызове функции. Здесь аргумент *`data`* — это переменная

с одним байтом данных. Вы можете послать несколько байтов, но для каждого придется вызвать `Wire.write()`:

```
Wire.write(data);
```

3. По окончании передачи данных определенному устройству завершите связь вызовом:

```
Wire.endTransmission();
```

Чтобы запросить данные из устройства на шине I²C, инициализируйте связь вызовом `Wire.beginTransmission(address)` и отправьте запрос:

```
Wire.requestFrom(address,x);
```

(где *x* — количество запрашиваемых байтов данных).

С помощью следующей функции сохраните каждый принятый байт в переменной:

```
incoming = Wire.read(); // incoming – переменная, куда
                        // сохраняется принятый байт данных
```

После приема завершите связь вызовом `Wire.endTransmission()`. Все эти функции мы используем в следующем проекте.

Проект 54: внешнее EEPROM

В главе 18 мы использовали EEPROM, встроенное в микроконтроллер на Arduino, чтобы обеспечить сохранность данных в случае сброса или отключения питания. Встроенное EEPROM способно хранить всего 1024 байта данных. Для хранения бóльшего объема нужно использовать внешние EEPROM, с которыми вы познакомитесь в этом проекте.

Рис. 19.4. Микросхема EEPROM Microchip Technology 24LC512

В качестве внешнего EEPROM мы используем микросхему 24LC512 от компании Microchip Technology, способную хранить 64 Кбайт (65 536 байт) данных (рис. 19.4). Ее можно приобрести у таких продавцов, как Digi-Key (артикул 24LC512-I/P-ND) и PMD Way (артикул 24LC512A).

Оборудование

Ниже перечислено оборудование для этого проекта:

- плата Arduino и кабель USB;
- одна микросхема EEPROM Microchip Technology 24LC512;

- одна макетная плата;
- два резистора номиналом 4,7 кОм;
- один керамический конденсатор емкостью 100 нФ;
- несколько отрезков провода разной длины.

Схема

Подключите через резисторы 4,7 кОм линии SCL и SDA к контакту 5 V, как на рис. 19.5.

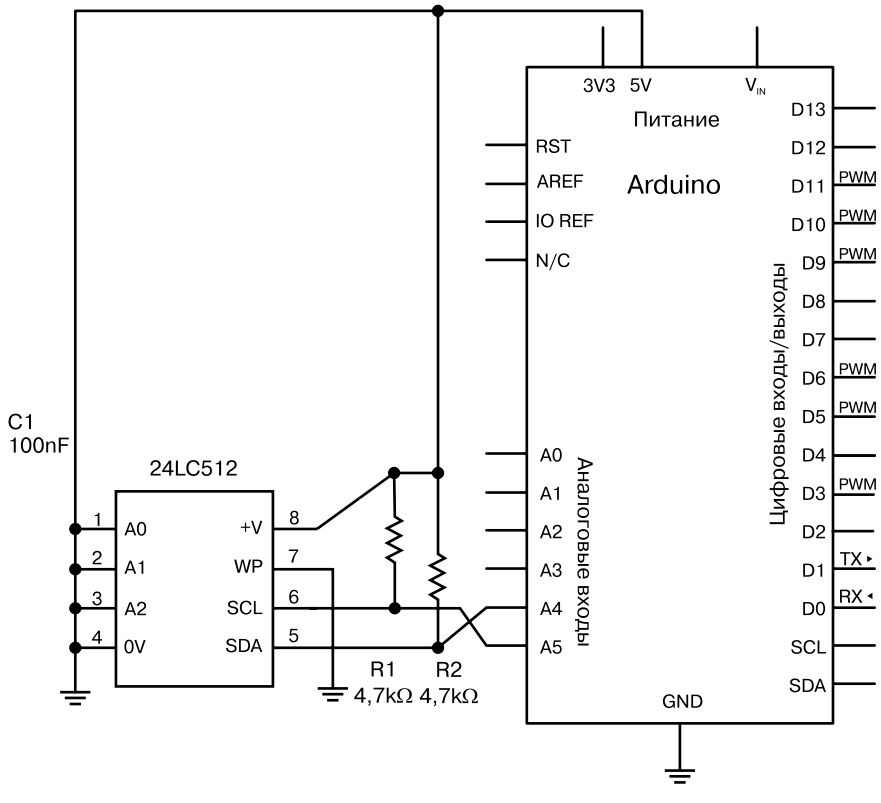


Рис. 19.5. Принципиальная схема проекта *54*

Адрес EEPROM 24LC512 на шине I²C частично определяется схемой подключения. Последние три бита адреса определяются состоянием выводов A2, A1 и A0 микросхемы. При подключении к «земле» они принимают значение 0, а при подключении к контакту 5 V — значение 1.

Первые четыре бита адреса предустановлены в состояние `1010`. То есть в этой схеме полный адрес EEPROM на шине I²C имеет вид `1010000` — в двоичном представлении или `0x50` — в шестнадцатеричном. Это значит, что в скетче мы должны использовать адрес `0x50`.

Скетч

Несмотря на то что внешнее EEPROM может хранить до 64 Кбайт данных, наш демонстрационный скетч будет сохранять и извлекать байты только из 20 первых ячеек в EEPROM.

Введите и загрузите следующий скетч:

```
  // Проект 54 — внешнее EEPROM
❶ #include <Wire.h>
  #define chip1 0x50

  byte d=0;

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

  void writeData(int device, unsigned int address, byte data)
  // Записывает байт данных 'data' в EEPROM с I2C-адресом 'device'
  // в ячейку с номером 'address'
  {
❸   Wire.beginTransmission(device);
    Wire.write((byte)(address >> 8));   // Старший байт номера ячейки
    Wire.write((byte)(address & 0xFF)); // и младший байт
    Wire.write(data);
    Wire.endTransmission();
    delay(10);
  }

❹ byte readData(int device, unsigned int address)
  // Читает байт данных из ячейки с номером 'address'
  // в EEPROM  с I2C-адресом 'device'
  {
    byte result;                        // Возвращаемое значение
    Wire.beginTransmission(device);
    Wire.write((byte)(address >> 8));   // Старший байт номера ячейки
    Wire.write((byte)(address & 0xFF)); // и младший байт
    Wire.endTransmission();
```

```
❺   Wire.requestFrom(device,1);
    result = Wire.read();
    return result;                        // Вернуть прочитанный байт
                                          // как результат функции readData
}

void loop()
{
  Serial.println("Writing data...");
  for (int a=0; a<20; a++)
  {
    writeData(chip1,a,a);
  }
  Serial.println("Reading data...");
  for (int a=0; a<20; a++)
  {
    Serial.print("EEPROM position ");
    Serial.print(a);
    Serial.print(" holds ");
    d=readData(chip1,a);
    Serial.println(d, DEC);
  }
}
```

Давайте пройдемся по скетчу. В ❶ выполняется подключение библиотеки и определяется адрес EEPROM на шине I²C в виде константы с именем `chip1`. В ❷ инициализируется монитор порта, а после — шина I²C. В скетч добавлены две пользовательских функции — `writeData()` и `readData()`. Они позволят сэкономить время и дадут вам код, который вы сможете использовать в будущих проектах для работы с устройствами EEPROM, поддерживающими подключение к шине I²C. Эти функции позволяют записывать данные в EEPROM и извлекать их.

Функция `writeData()` ❸ инициализирует связь с EEPROM, посылает номер ячейки для записи байта данных, дважды вызывая функцию `Wire.write()`, посылает байт данных для записи и завершает передачу.

Функция `readData()` ❹ действует похоже, но не посылает байт данных в EEPROM, а вызывает `Wire.requestFrom()`, чтобы прочитать данные ❺. Далее байт данных из EEPROM присваивается переменной `result` и возвращается как значение функции.

Результат

В функции `void loop()` скетч выполняет 20 итераций и записывает значения (от 0 до 19) в EEPROM. Дальше выполняются еще 20 итераций, в ходе которых извлекаются значения из 20 первых ячеек и выводятся в монитор порта (рис. 19.6).

```
Output  Serial Monitor ×
Message (Ctrl+Enter to send message to 'Arduino Uno' on 'COM6')
Writing data...
Reading data...
EEPROM position 0 holds 0
EEPROM position 1 holds 1
EEPROM position 2 holds 2
EEPROM position 3 holds 3
EEPROM position 4 holds 4
EEPROM position 5 holds 5
EEPROM position 6 holds 6
EEPROM position 7 holds 7
EEPROM position 8 holds 8
EEPROM position 9 holds 9
EEPROM position 10 holds 10
EEPROM position 11 holds 11
```

Рис. 19.6. Результаты работы проекта 54

Проект 55: расширитель цифровых портов

Расширитель порта — еще одно полезное устройство, подключаемое к шине I²C. Оно предназначено для увеличения числа цифровых входов/выходов. В этом проекте используется 16-битный расширитель портов Microchip Technology MCP23017 (рис. 19.7) с шестнадцатью цифровыми входами/выходами. Его можно приобрести у таких продавцов, как Digi-Key (артикул MCP23017-E/SP-ND) и PMD Way (артикул MCP23017A).

Рис. 19.7. Расширитель порта Microchip Technology MCP23017

В этом проекте мы подключим MCP23017 к плате Arduino и посмотрим, как управлять входами/выходами 16-битного расширителя порта. Все входы/выходы расширителя можно использовать подобно обычным цифровым входам/выходам Arduino.

Оборудование

Ниже перечислено оборудование для этого проекта:

- плата Arduino и кабель USB;
- одна макетная плата;
- несколько отрезков провода разной длины;

- одна микросхема расширителя порта Microchip Technology MCP20317;
- два резистора номиналом 4,7 кОм;
- (не обязательно) равное количество резисторов номиналом 560 Ом и светодиодов.

Схема

На рис. 19.8 изображена базовая схема подключения расширителя портов MCP23017. Так же как при подключении EEPROM в проекте 54, мы можем установить адрес устройства на шине I²C, подключив его выводы определенным образом. Подключите выводы MCP23017 с 15 по 17 к контакту GND, чтобы установить адрес `0x20`.

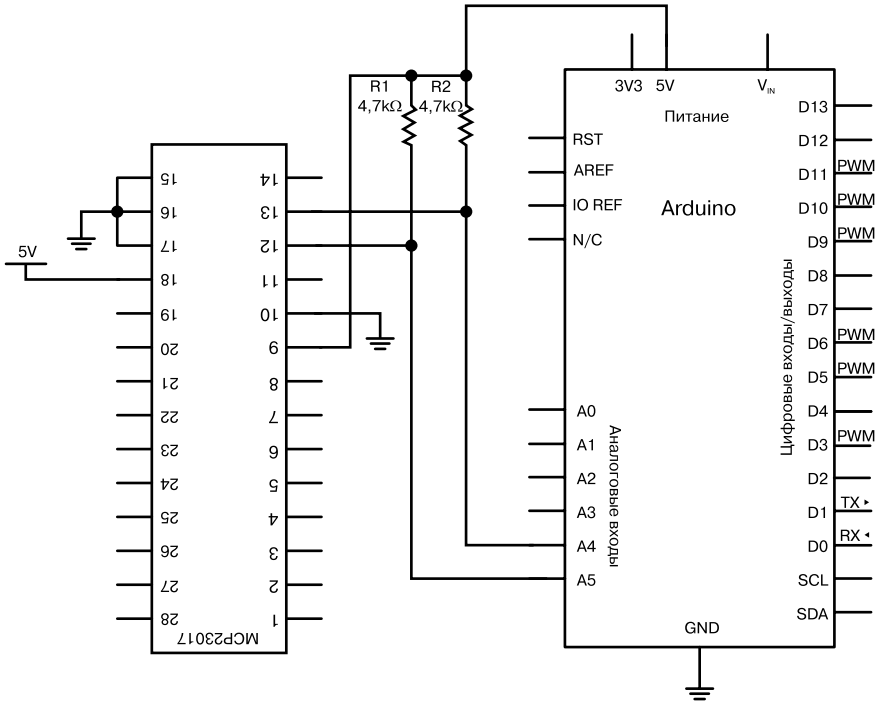


Рис. 19.8. Базовая принципиальная схема проекта *55*

При работе с MCP23017 лучше иметь перед глазами схему разводки выводов из документации с описанием, как на рис. 19.9. Обратите внимание, что 16 цифровых входов/выходов делятся на две группы: выводы с GPA7 по GPA0 находятся справа, а выводы с GPB0 по GPB7 — слева. Подключите светодиоды (через резисторы номиналом 560 Ом) ко всем или к выбранным выходам, чтобы видеть, когда на них устанавливается высокий уровень.

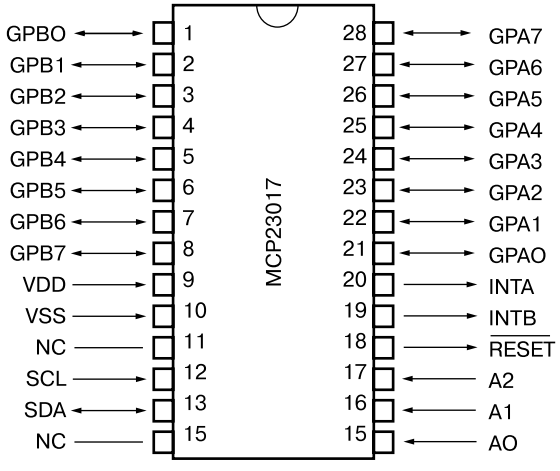


Рис. 19.9. Схема разводки выводов MCP23017

Скетч

Введите и загрузите следующий скетч:

```
// Проект 55 – расширитель порта

#include "Wire.h"
#define mcp23017 0x20

void setup()
{
❶  Wire.begin(); // Инициализировать шину I2C
   // Настроить входы/выходы MCP23017
   // на работу в режиме выходов
   Wire.beginTransmission(mcp23017);
   Wire.write(0x00); // Регистр IODIRA
   Wire.write(0x00); // Все выводы в группе A – выходы
   Wire.write(0x00); // Все выводы в группе B – выходы
❷  Wire.endTransmission();
}

void loop()
{
   Wire.beginTransmission(mcp23017);
   Wire.write(0x12);
❸  Wire.write(255); // Группа A
❹  Wire.write(255); // Группа B
   Wire.endTransmission();
   delay(1000);
   Wire.beginTransmission(mcp23017);
   Wire.write(0x12);
   Wire.write(0); // Группа A
```

```
    Wire.write(0); // Группа B
    Wire.endTransmission();
    delay(1000);
  }
```

Для работы с расширителем MCP23017 нужны все строки в функции `void setup()` с ❶ по ❷. Чтобы изменить состояние выходов в каждой группе, по очереди посылаются байты, представляющие эти группы. Сначала посылается байт, представляющий группу выходов с GPA0 по GPA7, а потом — представляющий группу выходов с GPB0 по GPB7.

Если понадобится изменить состояния отдельных выходов, рассматривайте каждую группу как двоичное число (это описывается в разделе «Краткое введение в двоичную систему счисления» в главе 6). Чтобы установить высокий уровень на выходах с 7 по 4, нужно послать двоичное число 11110000 (или десятичное 240), передав его функции `Wire.write()`, как показано в строке ❸, для группы GPA0–GPA7, или в строке ❹ для группы GPB0–GPB7.

Есть сотни устройств, поддерживающих взаимодействия по шине I²C. Теперь, зная, как работать с этой шиной, вы легко сможете подключать и использовать их с платой Arduino.

Шина SPI

В отличие от I²C шина SPI может применяться для одновременной передачи данных в обоих направлениях и с разными скоростями, в зависимости от типа используемого микроконтроллера. Но сами взаимодействия все так же осуществляются по схеме «ведущий/ведомый». Arduino играет роль ведущего устройства и определяет, с каким устройством (ведомым) она будет взаимодействовать.

Контакты

У каждого устройства, поддерживающего подключение к шине SPI, есть четыре линии для обмена данными: *MOSI* (Master-Out, Slave-In — «ведущий посылает, ведомый принимает»), *MISO* (Master-In, Slave-Out — «ведущий принимает, ведомый посылает»), SCK (тактовая линия) и SS, или CS, (Slave Select, или Chip Select, — «выбор ведомого», или «выбор устройства»). Эти линии подключаются к плате Arduino, как на рис. 19.10.

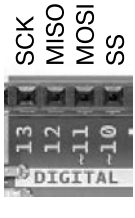


Рис. 19.10. Контакты на плате Arduino Uno, служащие одновременно выводами шины SPI

На рис. 19.11 показана схема типичного подключения устройства к плате Arduino через шину SPI. Контакты с D11 по D13 на плате зарезервированы для подключения линий

MISO, MOSI и SCK шины SPI. Но линию SS можно подключить к любому другому цифровому контакту (часто для этого используется контакт D10, так как он находится рядом с контактами, зарезервированными для шины SPI).

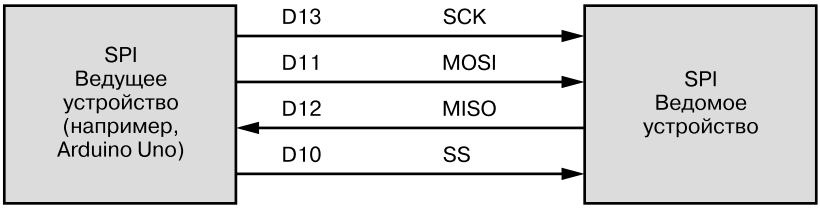


Рис. 19.11. Схема типичного подключения SPI-устройства к плате Arduino

ПРИМЕЧАНИЕ

Как и I^2C-устройства, SPI-устройства должны питаться напряжением 5 В или быть совместимы с таким напряжением, так как Arduino работает от 5 В. Обязательно уточняйте эту информацию перед покупкой.

Осуществление обмена данными по шине SPI

Теперь посмотрим, как осуществить в скетче обмен данными по шине SPI. Но прежде познакомимся с некоторыми функциями. Во-первых, к скетчу нужно подключить библиотеку SPI (входит в состав Arduino IDE):

```
#include "SPI.h"
```

Затем в функции `void setup()` настроить контакт для подключения линии SS на работу в режиме цифрового выхода. В примере ниже используется только одно SPI-устройство. Поэтому мы будем использовать контакт D10 и сразу же устанавливать на нем уровень `HIGH`, потому что большинство SPI-устройств переходят в активное состояние, когда на линии SS устанавливается низкий уровень:

```
pinMode(10, OUTPUT);
digitalWrite(10, HIGH);
```

Теперь нужно вызвать функцию инициализации шины SPI:

```
SPI.begin();
```

В конце скетч должен определить, как именно будут посылаться и приниматься данные. Некоторые SPI-устройства требуют, чтобы первым посылался старший бит (Most Significant Bit, MSB). Другие, наоборот, требуют, чтобы старший бит посылался последним (еще раз посетите раздел «Краткое введение в двоичную систему

счисления» в главе 6, где рассказывается о том, что такое старший бит). Поэтому внутри функции `void setup()`, сразу вслед за вызовом `SPI.begin()`, мы вызовем:

```
SPI.setBitOrder(order);
```

где аргумент *`order`* может иметь значение `MSBFIRST` или `MSBLAST`.

Передача данных SPI-устройству

Перед посылкой данных SPI-устройству нужно установить на линии SS уровень `LOW`, чтобы сообщить этому устройству, что оно выбрано ведущим (платой Arduino) для взаимодействий. Дальше нужно послать необходимое число байтов данных соответствующим числом вызовов следующей функции (то есть эту функцию придется вызвать для каждого байта):

```
SPI.transfer(byte);
```

По завершении взаимодействия с устройством установите уровень `HIGH` на линии SS, чтобы сообщить ему, что плата Arduino завершила сеанс связи.

Каждое SPI-устройство должно иметь отдельную линию SS. Например, если к шине SPI подключено два устройства, линию SS, соединяющую второе устройство, можно подключить к контакту D9 на плате Arduino (рис. 19.12).

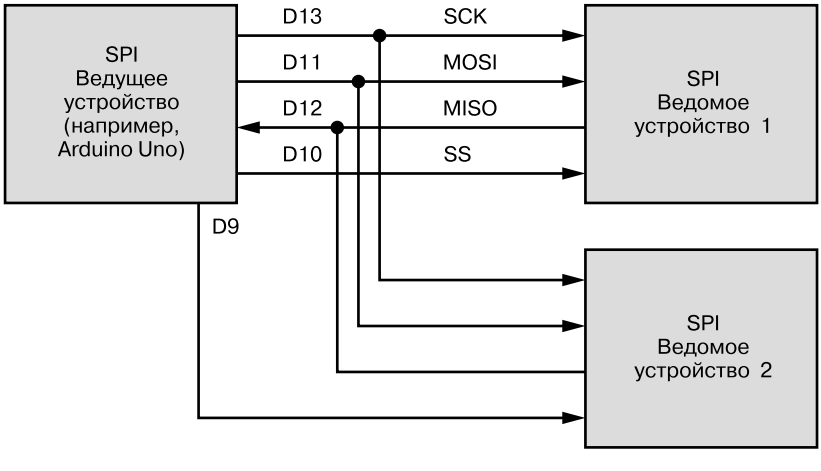


Рис. 19.12. Подключение двух SPI-устройств к плате Arduino

Для взаимодействий со вторым ведомым устройством до и после каждого сеанса связи используйте вывод D9 (вместо D10).

Проект 56 демонстрирует использование шины SPI для взаимодействий с цифровым реостатом.

Проект 56: цифровой реостат

Реостат (или потенциометр) — это устройство, похожее на резистор переменного сопротивления из главы 4. Разница только в том, что у реостата есть два вывода: один — это аналог среднего вывода переменного резистора, а второй — одного из его концов. В этом проекте мы будем менять сопротивление с помощью цифрового реостата вместо поворота ручки переменного резистора вручную.

Реостаты часто применяются для регулировки громкости в аудиооборудовании. Там вместо поворотных ручек используются кнопки, как, например, в автомобильных стерео. Допуск отклонений сопротивления реостата от номинала намного больше обычного постоянного сопротивления — иногда он доходит до 20 %.

В проекте 56 мы используем реостат Microchip Technology MCP4162 (рис. 19.13). Реостат MCP4162 выпускается с разными номиналами сопротивлений. Для нашего проекта выберем модель 10 кОм. Ее можно приобрести у таких продавцов, как Digi-Key (артикул MCP4162-103E/P-ND) и element14 (артикул 1840698). У выходного сопротивления 257 уровней регулировки с шагом около 40 Ом. Для выбора определенного уровня нужно послать два байта: байт команды (со значением 0) и байт значения (определяет уровень от 0 до 256). В реостате MCP4162 используется энергонезависимая память, поэтому после отключения и включения питания устанавливается последний выбранный уровень.

Рис. 19.13. Реостат Microchip Technology MCP4162

Мы используем реостат для управления яркостью свечения светодиода.

Оборудование

Ниже перечислено оборудование для этого проекта:

- плата Arduino и кабель USB;
- одна макетная плата;
- несколько отрезков провода разной длины;
- один цифровой реостат Microchip Technology MCP4162;
- один резистор номиналом 560 Ом;
- один светодиод.

Схема

На рис. 19.14 изображена принципиальная схема проекта 56. Нумерация выводов микросхемы MCP4162 начинается слева снизу. Вывод 1 обозначен точкой слева от логотипа Microchip (см. рис. 19.13).

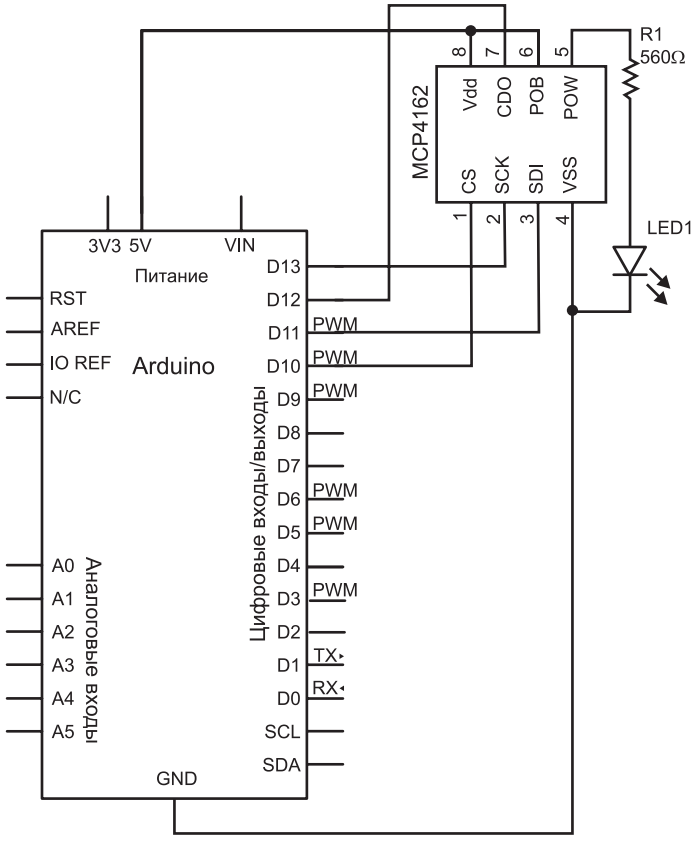


Рис. 19.14. Принципиальная схема проекта *56*

Скетч

Введите и загрузите следующий скетч:

```
  // Проект 56 — цифровой реостат
❶ #include "SPI.h" // Подключить библиотеку
  int ss=10;       // Использовать цифровой контакт 10
                   // для выбора ведомого SPI-устройства
  int del=200;     // Задержка между изменениями
                   // уровня яркости светодиода

  void setup()
  {
❷   SPI.begin();
    pinMode(ss, OUTPUT);    // Настройка вывода для подключения линии SS
    digitalWrite(ss, HIGH); // Выбор устройства происходит при низком
                            // уровне на линии SS, поэтому первоначально
                            // установить высокий уровень
```

```
❸   SPI.setBitOrder(MSBFIRST);
    // Реостат MCP4162 требует передавать биты данных в порядке
    // от старшего к младшему
  }

❹ void setValue(int value)
  {
    digitalWrite(ss, LOW);
    SPI.transfer(0);     // Послать байт команды
    SPI.transfer(value); // Послать значение (от 0 до 255)
    digitalWrite(ss, HIGH);
  }

  void loop()
  {
❺   for (int a=0; a<256; a++)
    {
      setValue(a);
      delay(del);
    }
❻   for (int a=255; a>=0; a--)
    {
      setValue(a);
      delay(del);
    }
  }
```

Теперь пройдемся по скетчу. Сначала он подключает библиотеку SPI и настраивает шину (❶ и ❷). В ❸ определяется порядок передачи данных, поддерживаемый реостатом MPC4162. Для упрощения регулировки сопротивления в скетче применяется пользовательская функция ❹. Она принимает уровень сопротивления (от 0 до 255) и посылает его в MCP4162. В конце скетч выполняет два цикла, в которых перебираются все уровни сопротивления, от нуля до максимума ❺ и обратно до нуля ❻. Этот последний фрагмент заставит светодиод увеличивать и уменьшать яркость свечения снова и снова, пока выполняется скетч.

Что дальше?

В этой главе вы поэкспериментировали с двумя важными способами взаимодействий, поддерживаемыми платой Arduino. Теперь вы сможете подключить к ней множество разных датчиков и других компонентов. Один из наиболее популярных на сегодняшний день — I^2C-часы реального времени, позволяющие проектам хранить текущее время и работать с функциями времени. Это и станет темой главы 20. Продолжим!

20

Часы реального времени

В этой главе вы:

- научитесь устанавливать и извлекать время и дату из модуля часов реального времени;
- познакомитесь с новыми способами подключения устройств к плате Arduino;
- сконструируете цифровые часы;
- сконструируете счетчик рабочего времени сотрудника, управляемый RFID-меткой.

I²C-модуль *часов реального времени* (Real-Time Clock, RTC) — это небольшое устройство-хронометр, открывающее новые возможности для проектов на основе Arduino. После установки времени и даты модуль RTC обеспечит высокую точность хода внутренних часов и возврат времени и даты по запросу.

На рынке есть много разных модулей часов реального времени с разной точностью измерения. В этой главе мы воспользуемся модулем Maxim DS3231. Для него не нужно ничего, кроме батарейки резервного питания. Этот модуль гарантирует высокую точность и надежность. Maxim DS3231 в форме небольшой платы с разъемом (рис. 20.1) можно приобрести у разных продавцов, например у PMD Way (артикул 883422).

Подключение модуля RTC

Подключить модуль RTC к плате Arduino просто. Для этого используется шина I²C (из главы 19). Берем четыре провода: выводы GND и VCC на плате модуля нужно соединить с контактами GND и 5 V на Arduino, а выводы SDA и SCL — с контактами A4 и A5. Остальные контакты у нас использоваться не будут. Конструктивные

особенности модуля избавляют от необходимости использовать «подтягивающие» резисторы для шины I²C.

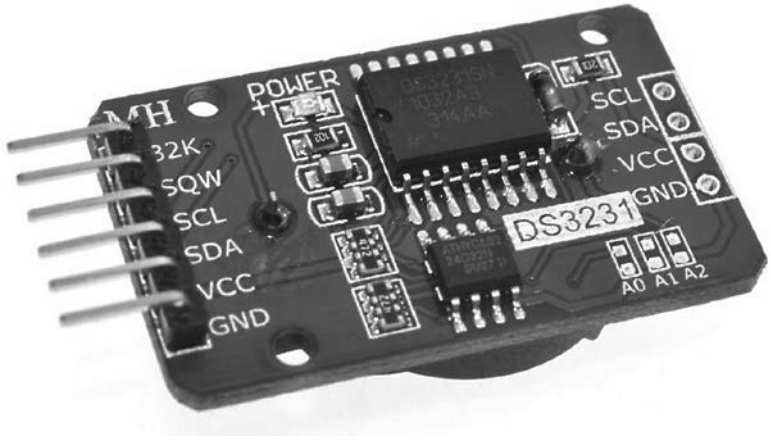


Рис. 20.1. I²C-модуль часов реального времени

Для удобства смонтируем модуль на макетной плате ProtoShield. Это упростит его интеграцию с другими электронными компонентами в дальнейшем. Не забудьте вставить батарейку резервного питания, иначе время будет сбрасываться при каждом выключении!

Проект 57: установка, отображение даты и времени

В этом проекте вы узнаете, как устанавливать дату и время в модуле RTC и как извлекать и отображать их в окне монитора порта. Знание даты и времени может пригодиться в самых разных проектах. Например в термометре с памятью или в будильнике.

Оборудование

Ниже перечислено оборудование для данного проекта:

- плата Arduino и кабель USB;
- несколько отрезков провода разной длины;
- одна батарея CR2032 battery (если не была приобретена вместе с модулем DS3231);
- модуль Maxim DS3231 RTC.

Скетч

Подключите модуль к плате Arduino, как описано выше, и введите, *но не загружайте* следующий скетч:

```
  // Проект 57 – установка и отображение даты и времени
❶ #include "Wire.h"

  #define DS3231_I2C_ADDRESS 0x68

  // Преобразует обычное десятичное число в двоично-десятичное
❷ byte decToBcd(byte val)
  {
    return( (val/10*16) + (val%10) );
  }

  // Преобразует двоично-десятичное число в обычное десятичное
  byte bcdToDec(byte val)
  {
    return( (val/16*10) + (val%16) );
  }

❸ void setDS3231time(byte second, byte minute, byte hour,
                     byte dayOfWeek, byte dayOfMonth,
                     byte month, byte year)
  {
    // Записать дату и время в DS3231
    Wire.beginTransmission(DS3231_I2C_ADDRESS);
    Wire.write(0); // Выбрать для записи регистр секунд
    Wire.write(decToBcd(second));     // Записать секунды
    Wire.write(decToBcd(minute));     // Записать минуты
    Wire.write(decToBcd(hour));       // Записать часы
    Wire.write(decToBcd(dayOfWeek));  // Записать день недели (1=воскресенье,
                                      // 7=суббота)
    Wire.write(decToBcd(dayOfMonth)); // Записать число месяца (от 1 до 31)
    Wire.write(decToBcd(month));      // Записать месяц
    Wire.write(decToBcd(year));       // Записать год (от 0 до 99)
    Wire.endTransmission();
  }

❹ void readDS3231time(byte *second, byte *minute, byte *hour,
                      byte *dayOfWeek, byte *dayOfMonth,
                      byte *month, byte *year)
  {
    Wire.beginTransmission(DS3231_I2C_ADDRESS);
    Wire.write(0); // Инициализировать регистр указателя в DS3231
    Wire.endTransmission();
    Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
```

```
  // Прочитать семь байт данных из DS3231, начиная с регистра 00h
  *second     = bcdToDec(Wire.read() & 0x7f);
  *minute     = bcdToDec(Wire.read());
  *hour       = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek  = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month      = bcdToDec(Wire.read());
  *year       = bcdToDec(Wire.read());
}

void displayTime()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;

  // Прочитать данные из DS3231
❺ readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth,
                 &month, &year);

  // Вывести в монитор порта
  Serial.print(hour, DEC); // Каждый байт выводить как десятичное число
  Serial.print(":");
  if (minute<10)
  {
    Serial.print("0");
  }
  Serial.print(minute, DEC);
  Serial.print(":");
  if (second<10)
  {
    Serial.print("0");
  }
  Serial.print(second, DEC);
  Serial.print(" ");
  Serial.print(dayOfMonth, DEC);
  Serial.print("/");
  Serial.print(month, DEC);
  Serial.print("/");
  Serial.print(year, DEC);
  Serial.print(" Day of week: ");
  switch(dayOfWeek){
  case 1:
    Serial.println("Sunday");
    break;
  case 2:
    Serial.println("Monday");
    break;
  case 3:
    Serial.println("Tuesday");
    break;
```

```
    case 4:
      Serial.println("Wednesday");
      break;
    case 5:
      Serial.println("Thursday");
      break;
    case 6:
      Serial.println("Friday");
      break;
    case 7:
      Serial.println("Saturday");
      break;
    }
  }

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

    // Установить начальное время:
    // секунды, минуты, часы, день недели, число, месяц, год
❻   setDS3231time(0, 56, 23, 6, 30, 10, 21);
  }

 void loop()
  {
    displayTime(); // Выводить текущее время в монитор порта
    delay(1000);   // каждую секунду
  }
```

Принцип действия

Скетч может показаться сложным, но это не так. В ❶ мы импортируем библиотеку I2C и устанавливаем адрес шины RTC в скетче как `0x68`. Это адрес по умолчанию для DS3231, как определено в документации. В ❷ объявляются две пользовательские функции для преобразования десятичных чисел в двоично-десятичные (BCD) и обратно. Это важно, так как DS3231 хранит значения в двоично-десятичном формате.

В строке ❻ вызывается функция `setDS3231time` для записи времени и даты в модуль RTC, которая имеет следующий синтаксис:

```
setDS3231time(second, minute, hour, dayOfWeek, dayOfMonth, month, year)
```

Чтобы воспользоваться ею, просто подставьте необходимые данные вместо параметров. Параметр *`dayOfWeek`* — это число от 1 до 7, представляющее день недели

(от воскресенья до субботы). У RTC нет возможности проверять соответствие дня недели введенной дате, поэтому будьте внимательны при вводе каждого параметра. Параметр *year* интерпретируется как двузначное число. Например, 2021 году соответствует число `21` (начальное 20 подразумевается). В параметрах можно передавать фиксированные значения (как в скетче) или переменные типа `byte`.

Запись даты и времени в модуль RTC осуществляет функция `setDS3231time` ❸. Теперь загрузите скетч. После этого закомментируйте вызов функции, добавив пару символов `//` в начало строки ❻ перед именем `setDS3231time`, и загрузите скетч повторно, чтобы время не устанавливалось каждый раз в одно и то же значение после каждого запуска скетча!

Наконец, функция `readDS3231time` ❹ читает дату и время из модуля RTC и сохраняет их в переменных типа `byte`. Она вызывается в строке ❺, внутри функции `displayTime`, которая извлекает данные и выводит их в монитор порта.

Теперь загрузите скетч и откройте окно монитора порта. Результаты должны выглядеть примерно так (рис. 20.2).

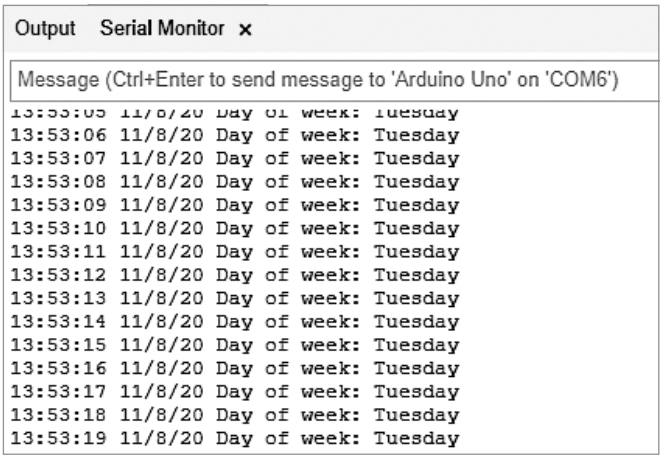


Рис. 20.2. Результаты работы проекта *57*

Скетч для проекта 57 можно взять как основу для других проектов, использующих дату и время. Можно вставлять функции `decToBcd`, `bcdToDec`, `readDS3231time` и `setDS3231time` и повторно их использовать. Это одно из достоинств платформы Arduino: однажды написанная процедура может использоваться в других проектах с небольшими изменениями или вообще без них.

Проект 58: простые цифровые часы

Здесь мы воспользуемся функциями из проекта 57 для отображения времени и даты на экране стандартного ЖКИ. С таким экраном мы имели дело в проекте 44 (простой приемник GPS) в главе 15.

Оборудование

Ниже перечислено оборудование для данного проекта:

- плата Arduino и кабель USB;
- несколько отрезков провода разной длины;
- одна макетная плата;
- одна макетная плата ProtoScrewShield или подобная ей;
- модуль или плата расширения с ЖКИ;
- модуль часов реального времени (показанный выше в этой главе).

Сначала воссоздадим устройство из проекта 57. Если до этого вы подключали модуль RTC к Arduino проводами, теперь для той же цели воспользуйтесь платой ProtoScrewShield. Затем вставьте плату расширения LCD поверх других.

Скетч

Введите и загрузите следующий скетч:

```
// Проект 58 – простые цифровые часы

#include "Wire.h"
❶ #include <LiquidCrystal.h>
#define DS3231_I2C_ADDRESS 0x68

LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );

// Преобразует обычное десятичное число в двоично-десятичное
byte decToBcd(byte val)
{
  return( (val/10*16) + (val%10) );
}

// Преобразует двоично-десятичное число в обычное десятичное
byte bcdToDec(byte val)
{
  return( (val/16*10) + (val%16) );
}
```

```
void setDS3231time(byte second, byte minute, byte hour,
                   byte dayOfWeek, byte dayOfMonth,
                   byte month, byte year)
{
  // Записать дату и время в DS3231
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // Выбрать для записи регистр секунд
  Wire.write(decToBcd(second));     // Записать секунды
  Wire.write(decToBcd(minute));     // Записать минуты
  Wire.write(decToBcd(hour));       // Записать часы
  Wire.write(decToBcd(dayOfWeek));  // Записать день недели (1=воскресенье,
                                    // 7=суббота)
  Wire.write(decToBcd(dayOfMonth)); // Записать число месяца (от 1 до 31)
  Wire.write(decToBcd(month));      // Записать месяц
  Wire.write(decToBcd(year));       // Записать год (от 0 до 99)
  Wire.endTransmission();
}

void readDS3231time(byte *second, byte *minute, byte *hour,
                    byte *dayOfWeek, byte *dayOfMonth,
                    byte *month, byte *year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // Инициализировать регистр указателя в DS3231
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);

  // Прочитать семь байт данных из DS3231, начиная с регистра 00h
  *second     = bcdToDec(Wire.read() & 0x7f);
  *minute     = bcdToDec(Wire.read());
  *hour       = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek  = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month      = bcdToDec(Wire.read());
  *year       = bcdToDec(Wire.read());
}

void displayTime()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;

  // Прочитать данные из DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth,
                 &month, &year);

  // Передать данные в плату расширения LCD
  lcd.clear();
  lcd.setCursor(4,0);
  lcd.print(hour, DEC);
  lcd.print(":");
  if (minute<10)
```

```
  {
    lcd.print("0");
  }
  lcd.print(minute, DEC);
  lcd.print(":");
  if (second<10)
  {
    lcd.print("0");
  }
  lcd.print(second, DEC);

  lcd.setCursor(0,1);
  switch(dayOfWeek){
  case 1:
    lcd.print("Sun");
    break;
  case 2:
    lcd.print("Mon");
    break;
  case 3:
    lcd.print("Tue");
    break;
  case 4:
    lcd.print("Wed");
    break;
  case 5:
    lcd.print("Thu");
    break;
  case 6:
    lcd.print("Fri");
    break;
  case 7:
    lcd.print("Sat");
    break;
  }
  lcd.print(" ");
  lcd.print(dayOfMonth, DEC);
  lcd.print("/");
  lcd.print(month, DEC);
  lcd.print("/");
  lcd.print(year, DEC);
}

void setup()
{
  Wire.begin();
❷ lcd.begin(16, 2);
  // Установить начальное время:
  // секунды, минуты, часы, день недели, число, месяц, год
❸ setDS3231time(0, 56, 23, 6, 30, 10, 21);
}
```

```
void loop()
{
  displayTime(); // Отображать время на экране ЖКИ
  delay(1000);   // каждую секунду
}
```

Принцип действия и результаты

По действию этот скетч похож на тот, что мы использовали в проекте 57. Разница в том, что функция `displayTime()` выводит время не в монитор порта, а на экран ЖКИ. Сам же скетч включает строки, выполняющие настройку платы расширения с ЖКИ (❶ и ❷) (чтобы освежить в памяти приемы работы с модулем ЖКИ, обращайтесь к главе 9).

Перед первой загрузкой скетча не забудьте раскомментировать строку ❸, устанавливающую начальное время, а затем закомментировать ее и загрузить скетч повторно. После загрузки скетча вы увидите результаты на экране ЖКИ (рис. 20.3).

Рис. 20.3. Отображение даты и времени в проекте 58

Опыт, приобретенный в проектах 57 и 58, должен дать вам полное представление о том, как записать данные в часы реального времени и прочитать их оттуда. Теперь применим наши знания для создания чего-нибудь действительно полезного.

Проект 59: система хронометража с RFID-метками

В этом проекте мы создадим устройство для учета рабочего времени. Вы увидите, как можно использовать сразу несколько плат расширения и как ProtoScrewShield помогает подключать дополнительные компоненты, не смонтированные на плате расширения. Это устройство может читать карты RFID и засекать время прихода или ухода владельца карты (например, прихода на рабочее место и ухода домой). Время и числовой идентификатор карты будут записываться на microSD для дальнейшего анализа.

Мы уже видели, как записывать данные на карту microSD в главе 15, как работать с радиомаркерами RFID — в главе 18 и как пользоваться часами реального времени — в этой главе выше. Теперь объединим все это.

Оборудование

Ниже перечислено оборудование для этого проекта:

- плата Arduino и кабель USB;
- несколько отрезков провода разной длины;
- модуль RTC (выше в этой главе);
- модуль ЖКИ или плата расширения Freetronics ЖКИ;
- плата расширения с картой памяти MicroSD (из главы 15);
- одна макетная плата ProtoScrewShield или подобная;
- модуль RFID и два радиомаркера (из главы 18).

Начнем сборку устройства. Поместим Arduino Uno вниз и подключим к ней сверху платы расширения ProtoScrewShield, microSD и ЖКИ. Подключите модуль чтения RFID, как описывалось в главе 18, и модуль RTC, как описывалось выше в этой главе. В зависимости от моделей используемого оборудования собранное устройство может выглядеть как на рис. 20.4.

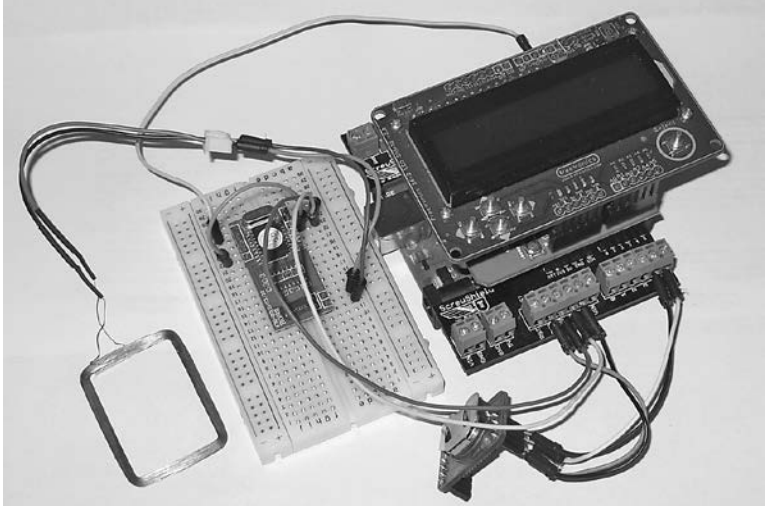


Рис. 20.4. Собранное устройство для хронометража

Скетч

Теперь введите и загрузите следующий скетч. Перед загрузкой не забудьте отсоединить провод, связывающий контакт RX модуля RFID с контактом D0 на плате Arduino, и восстановить соединение после успешной загрузки скетча.

```
// Проект 59 — система хронометража с радиомаркерами
❶ #include "Wire.h" // Для модуля RTC
❷ #include "SD.h"   // Для платы расширения с картой microSD
#include <LiquidCrystal.h>

#define DS3231_I2C_ADDRESS 0x68

LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );
int data1 = 0;

❸ // Для определения числовых идентификаторов радиомаркеров
// используйте листинг 18.1
int Mary[14] = {
  2, 52, 48, 48, 48, 56, 54, 67, 54, 54, 66, 54, 66, 3};
int John[14] = {
  2, 52, 48, 48, 48, 56, 54, 66, 49, 52, 70, 51, 56, 3};
int newtag[14] = {
  0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // Используется для чтения и сравнения

// Преобразует обычное десятичное число в двоично-десятичное
byte decToBcd(byte val)
{
  return( (val/10*16) + (val%10) );
}

// Преобразует двоично-десятичное число в обычное десятичное
byte bcdToDec(byte val)
{
  return( (val/16*10) + (val%16) );
}

void setDS3231time(byte second, byte minute, byte hour,
                   byte dayOfWeek, byte dayOfMonth,
                   byte month, byte year)
{
  // Записать дату и время в DS3231
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // Выбрать для записи регистр секунд
  Wire.write(decToBcd(second));     // Записать секунды
  Wire.write(decToBcd(minute));     // Записать минуты
  Wire.write(decToBcd(hour));       // Записать часы
  Wire.write(decToBcd(dayOfWeek));  // Записать день недели (1=воскресенье,
                                    // 7=суббота)
  Wire.write(decToBcd(dayOfMonth)); // Записать число месяца (от 1 до 31)
  Wire.write(decToBcd(month));      // Записать месяц
  Wire.write(decToBcd(year));       // Записать год (от 0 до 99)
  Wire.endTransmission();
}

void readDS3231time(byte *second, byte *minute, byte *hour,
                    byte *dayOfWeek, byte *dayOfMonth,
                    byte *month, byte *year)
```

```
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // Инициализировать регистр указателя в DS3231
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);

  // Прочитать семь байт данных из DS3231, начиная с регистра 00h
  *second     = bcdToDec(Wire.read() & 0x7f);
  *minute     = bcdToDec(Wire.read());
  *hour       = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek  = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month      = bcdToDec(Wire.read());
  *year       = bcdToDec(Wire.read());
}

// Сравнивает два массива и возвращает true, если идентичны
// Удобно использовать для сравнения числовых идентификаторов
// радиомаркеров
boolean comparetag(int aa[14], int bb[14])
{
  boolean ff=false;
  int fg=0;
  for (int cc=0; cc<14; cc++)
  {
    if (aa[cc]==bb[cc])
    {
      fg++;
    }
  }
  if (fg==14)
  {
    ff=true; // Все 14 элементов массивов совпадают
  }
  return ff;
}

void wipeNewTag()
{
  for (int i=0; i<=14; i++)
  {
    newtag[i]=0;
  }
}

void setup()
{
  Serial.flush(); // Очистить буфер последовательного порта
  Serial.begin(9600);
  Wire.begin();
  lcd.begin(16, 2);
```

```
  // Установить начальное время:
  // секунды, минуты, часы, день недели, число, месяц, год
  //setDS3231time(0, 56, 23, 6, 30, 10, 21);

  // Проверить наличие и готовность карты microSD
❹ if (!SD.begin(8))
  {
    lcd.print("uSD card failure");
    // Остановить скетч
    return;
  }
  lcd.print("uSD card OK");
  delay(1000);
  lcd.clear();
}

void loop()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;

  if (Serial.available() > 0) // Если была попытка чтения
  {
    // Прочитать число из модуля RFID
    delay(100); // Дать время поступить всем данным
                // в буфер последовательного порта
    for (int z=0; z<14; z++) // Прочитать числовой идентификатор радиомаркера
    {
      data1=Serial.read();
      newtag[z]=data1;
    }
    Serial.flush(); // Аннулировать повторные попытки чтения

    // Извлечь данные из модуля DS3231
    readDS3231time(&second, &minute, &hour, &dayOfWeek,
                   &dayOfMonth, &month, &year);
  }

  // Выполнить необходимые операции по результатам
❺ if (comparetag(newtag, Mary) == true)
  {
    lcd.print("Hello Mary ");
    File dataFile = SD.open("DATA.TXT", FILE_WRITE);
    if (dataFile)
    {
      dataFile.print("Mary ");
      dataFile.print(hour);
      dataFile.print(":");
      if (minute<10) { dataFile.print("0"); }
      dataFile.print(minute);
      dataFile.print(":");
      if (second<10) { dataFile.print("0"); }
```

```
      dataFile.print(second);
      dataFile.print(" ");
      dataFile.print(dayOfMonth);
      dataFile.print("/");
      dataFile.print(month);
      dataFile.print("/");
      dataFile.print(year);
      dataFile.println();
      dataFile.close();
    }
    delay(1000);
    lcd.clear();
    wipeNewTag();
  }

  if (comparetag(newtag, John)==true)
  {
    lcd.print("Hello John ");
    File dataFile = SD.open("DATA.TXT", FILE_WRITE);
    if (dataFile)
    {
      dataFile.print("John ");
      dataFile.print(hour);
      dataFile.print(":");
      if (minute<10) { dataFile.print("0"); }
      dataFile.print(minute);
      dataFile.print(":");
      if (second<10) { dataFile.print("0"); }
      dataFile.print(second);
      dataFile.print(" ");
      dataFile.print(dayOfMonth);
      dataFile.print("/");
      dataFile.print(month);
      dataFile.print("/");
      dataFile.print(year);
      dataFile.println();
      dataFile.close();
    }
    delay(1000);
    lcd.clear();
    wipeNewTag();
  }
}
```

Принцип действия

В этом скетче система сначала ждет, пока карта RFID окажется в зоне действия антенны. Если карта опознана, в конец текстового файла на microSD записываются имя владельца, время и дата.

В строке ❶ подключается библиотека с функциями для работы с шиной I²C и часами реального времени. В ❷ подключается библиотека для работы с картой microSD. В строке ❹ проверяется и выводится информация о состоянии карты microSD. В ❺ только что прочитанный числовой идентификатор карты RFID сравнивается с известными идентификаторами двух людей — в данном случае Джона (John) и Мэри (Mary). В случае совпадения с одним из них данные записываются на microSD. Вы можете добавить в систему еще карты, просто вставив числовые идентификаторы ниже уже имеющихся ❸ и добавив инструкцию сравнения, как в строке ❺.

Когда придет время анализировать данные, скопируйте файл `data.txt` с карты microSD и откройте его в текстовом редакторе или импортируйте в электронную таблицу для более детального изучения. Данные отформатированы так, что легко читаются (рис. 20.5).

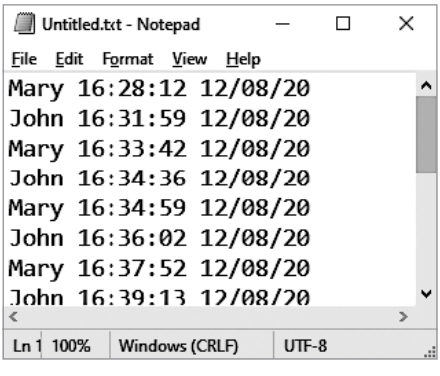


Рис. 20.5. Пример данных, накопленных устройством из проекта 59

Что дальше?

В этой главе вы узнали, как работать с датой и временем через I²C-модуль RTC. Система хронометража с радиомаркерами RFID из проекта 59 может стать основой для ваших собственных систем доступа. Она даже может следить за тем, когда ваши дети приходят домой. В двух последних главах мы создадим проекты на основе Arduino, где реализуем обмен данными через интернет и сети сотовой связи.

21

Интернет

В этой главе вы:

- создадите веб-сервер для отображения данных на веб-странице;
- научитесь с помощью Arduino посылать сообщения в Twitter;
- узнаете, как организовать дистанционное управление цифровыми выходами на плате Arduino из браузера.

В этой главе вы узнаете, как связать плату Arduino с внешним миром через интернет. Благодаря этой связи вы сможете рассылать данные из Arduino и дистанционно управлять ею из браузера.

Оборудование

Для создания связанных с интернетом проектов вам понадобятся: коммуникационное оборудование, кабель и кое-какая информация.

Начнем с оборудования. Вам потребуется плата расширения Ethernet с микросхемой контроллера W5100. Есть два варианта: оригинальная плата расширения Ethernet, выпускаемая Arduino (рис. 21.1), или плата, совместимая с Arduino Uno, включающая интегрированное оборудование Ethernet, например плата с артикулом 328497 у продавца PMD Way (рис. 21.2). Последняя отлично подходит для новых проектов и в тех случаях, когда есть ограничения по физическому объему для размещения оборудования или по финансовым соображениям. На рис. 21.2 у платы с интегрированным оборудованием Ethernet есть разъемы для подключения дополнительных плат расширения, порт USB, разъем Ethernet и гнездо для подключения карты памяти microSD.

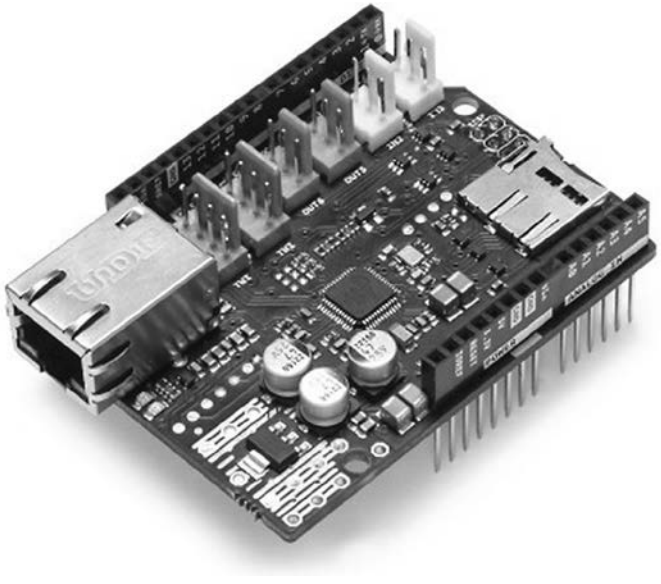


Рис. 21.1. Плата расширения Arduino Ethernet

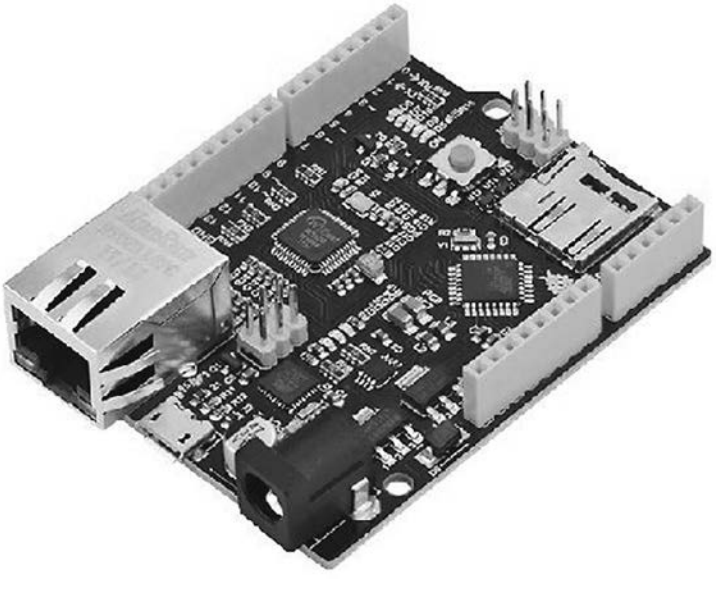


Рис. 21.2. Плата, совместимая с Arduino Uno
и интегрированным оборудованием Ethernet

Независимо от выбранного оборудования вам нужен стандартный сетевой кабель 10/100 CAT5, CAT5E или CAT6 для подключения платы расширения Ethernet к сетевому маршрутизатору или модему.

Еще важно знать IP-адрес вашего сетевого маршрутизатора или модема, который должен выглядеть примерно так: 192.168.0.1, и IP-адрес вашего компьютера, выглядящий так же, как и IP-адрес маршрутизатора.

Для обмена информацией с платой Arduino за пределами домашней или локальной сети вам нужен статический и общедоступный IP-адрес. *Статическим* называют фиксированный IP-адрес, присвоенный подключению к интернету вашим интернет-провайдером (Internet Service Provider, ISP). У подключения может не быть статического IP-адреса. Чтобы получить его, нужно связаться с поставщиком услуг. Если он не может предложить статический IP-адрес или стоимость аренды слишком высока, вы можете использовать автоматическую переадресацию с IP-адреса вашего подключения в сторонней компании, такой как No-IP (http://www.noip.com/) или Dyn (http://dyn.com/dns/).

А теперь проверим наше оборудование на примере простого проекта.

Проект 60: станция удаленного мониторинга

В проектах из предыдущих глав мы собирали данные с датчиков, чтобы оценить температуру и освещенность. Здесь вы узнаете, как отображать такие значения на простой веб-странице, которую можно просмотреть на любом устройстве, поддерживающем выход в интернет. Этот проект отображает уровни напряжений на аналоговых входах и состояния цифровых входов с нулевого по девятый на простой веб-странице. Это основа для создания устройства дистанционного мониторинга.

Позднее вы сможете подключить к этим входам дополнительные датчики с аналоговыми и цифровыми выходами (например, датчики температуры, освещенности и контактные датчики) и отображать их состояния на веб-странице.

Оборудование

Ниже перечислено оборудование для этого проекта:

- один кабель USB;
- один сетевой кабель;
- одна плата Arduino Uno с платой расширения Ethernet или плата, совместимая с Arduino Uno и имеющая интегрированное оборудование Ethernet.

Скетч

Введите следующий скетч, *но пока не загружайте его*:

```
/* Проект 60 – станция дистанционного мониторинга,
создан 18 декабря 2009 года Дэвидом Меллисом (David A. Mellis),
изменен 9 апреля 2012 года Томом Иго (Tom Igoe),
изменен в августе 2020 года Джоном Бокселлом (John Boxall)
*/

#include <SPI.h>
#include <Ethernet.h>

❶ IPAddress ip(xxx,xxx,xxx,xxx); // Замените IP-адресом своего проекта
❷ byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
EthernetServer server(80);

void setup()
{
  // Запустить сервер, обслуживающий соединение Ethernet
  Ethernet.begin(mac, ip);
  server.begin();
  for (int z=0; z<10; z++)
  {
    pinMode(z, INPUT); // Настроить цифровые контакты 0–9
                       // на работу в режиме входов
  }
}

void loop()
{
  // Ждать запросов от клиентов (на получение веб-страницы)
  EthernetClient client = server.available();
  if (client) {
    // http-запрос должен заканчиваться пустой строкой
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          // Добавить метатег Refresh, чтобы браузер
          // обновлял страницу каждые 5 с:
❸         client.println("<meta http-equiv=\"refresh\" content=\"5\">");
          // Добавить на страницу значение каждого аналогового входа
          for (int analogChannel = 0; analogChannel < 6; analogChannel++)
          {
            int sensorReading = analogRead(analogChannel);
```

```
❹             client.print("analog input ");
              client.print(analogChannel);
              client.print(" is ");
              client.print(sensorReading);
              client.println("<br />");
            }
            // Добавить на страницу значения цифровых входов 0–9
            for (int digitalChannel = 0; digitalChannel < 10; digitalChannel++)
            {
              boolean pinStatus = digitalRead(digitalChannel);
              client.print("digital pin ");
              client.print(digitalChannel);
              client.print(" is ");
              client.print(pinStatus);
              client.println("<br />");
            }
            client.println("</html>");
            break;
          }
          if (c == '\n') {
            // Начало новой строки
            currentLineIsBlank = true;
          }
          else if (c != '\r') {
            // В текущей строке имеются какие-то символы
            currentLineIsBlank = false;
          }
        }
      }
      // Дать время браузеру получить данные
      delay(1);
      // Закрыть соединение:
      client.stop();
    }
  }
```

Мы подробно обсудим этот скетч чуть позже. Прежде чем загрузить его, введите IP-адрес для своей платы расширения Ethernet, чтобы потом ее можно было найти в локальной сети. Первые три компонента адреса можно взять из IP-адреса вашего маршрутизатора. Например, если его адрес 192.168.0.1, подставьте вместо последнего числа другое случайное от 1 до 254, чтобы получился IP-адрес, не используемый никаким другим устройством в вашей сети. Введите готовый адрес в строке ❶ в скетче:

```
IPAddress ip(192, 168, 0, 69); // IP-адрес платы расширения Ethernet
```

После этого сохраните и загрузите скетч. Затем вставьте плату расширения Ethernet в разъемы на Arduino, соедините сетевым кабелем свой маршрутизатор или модем с платой расширения Ethernet и включите питание платы Arduino.

Подождите примерно 20 секунд, а после в браузере на любом устройстве в вашей сети введите IP-адрес из строки ❶. Если в браузере появится страница как на рис. 21.3, значит, основа станции дистанционного мониторинга работает правильно.

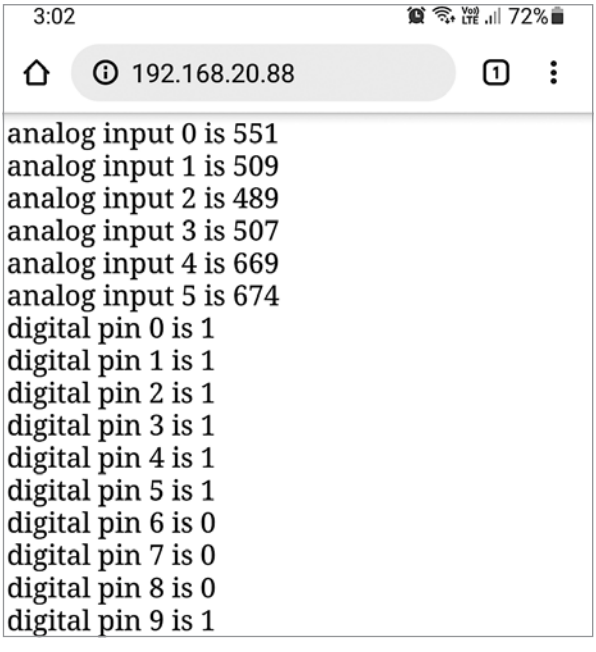


Рис. 21.3. Значения на аналоговых входах и состояние цифровых входов отображаются в виде веб-страницы на любом устройстве с браузером

Поиск и устранение неисправностей

Если проект не заработал, выполните следующие действия:

- проверьте правильность IP-адреса в строке ❶;
- проверьте, верно ли введен скетч и успешно ли он загружен;
- дважды проверьте подключение к локальной сети. Узнайте, есть ли у вашего компьютера выход в интернет, проверьте питание платы Arduino и ее подключение к маршрутизатору и модему;
- если вы пытаетесь получить доступ к веб-странице проекта со смартфона, убедитесь, что у него есть доступ к вашей локальной сети Wi-Fi;
- если на плате расширения Ethernet не мигает ни один светодиод после включения питания и подключения кабеля Ethernet к ней и к маршрутизатору или модему, то попробуйте заменить кабель.

Принцип действия

Если станция мониторинга работает, можно вернуться к самым важным участкам в скетче. Код от начала скетча и до строки ❸ обязательный — он импортирует необходимые библиотеки и активирует оборудование Ethernet, которое настраивается и запускается в функции `void setup()`. С помощью инструкций `client.print()` перед строкой ❸ скетч подготавливает веб-страницу для передачи браузеру. Начиная со строки ❸, функции `client.print()` и `client.println()` используются для вывода информации на веб-страницу, подобно тому как она выводится в монитор порта. Например, следующий код используется для вывода первых шести строк на веб-странице на рис. 21.3:

```
client.print("analog input ");
client.print(analogChannel);
client.print(" is ");
client.print(sensorReading);
```

В строке ❹ показан пример вывода текста и содержимого переменной на веб-странице. Здесь можно использовать разметку HTML для управления внешним видом страницы, при условии что память в Arduino не будет переполнена. Иначе говоря, вы можете использовать любой объем разметки HTML, пока размер скетча не превысит максимально допустимое значение, ограниченное объемом памяти на плате Arduino (объемы памяти каждого типа есть в табл. 13.2).

Еще один важный элемент скетча — это MAC-адрес. Он используется для обнаружения отдельных устройств, подключенных к сети. У каждого подключаемого к сети устройства есть уникальный MAC-адрес, который можно изменить подменой одного из шестнадцатеричных чисел в строке ❷. Если в одной сети одновременно должно быть несколько проектов на основе Arduino, введите разные MAC-адреса для каждого устройства.

Если вы захотите посмотреть веб-страницу на устройстве без доступа к локальной сети (на планшетном компьютере или телефоне, использующем сети сотовой связи), вам нужно настроить *переадресацию портов* (port forwarding) на своем маршрутизаторе или модеме с общедоступным IP-адресом, настроенным организациями вроде No-IP или Dyn.

Настройка переадресации портов в разных моделях маршрутизаторов выполняется по-разному. Поэтому ищите инструкции в Интернете или посетите сайт http://www.wikihow.com/Port-Forward/ для получения дополнительной информации.

Теперь, когда вы знаете, как отображать текст и значения переменных в виде веб-страницы, попробуем с помощью Arduino посылать сообщения в Twitter.

Проект 61: Arduino Tweeter

В этом проекте вы узнаете, как с помощью Arduino посылать сообщения в Twitter. Это поможет научиться принимать все виды информации с любых устройств, доступных скетчу, и посылать их в Twitter. Если вы захотите получать ежечасные замеры температуры в доме, находясь где-нибудь за границей, или извещения о том, когда дети приходят домой, в вашем распоряжении будет недорогое решение.

Для начала нужно будет создать для платы Arduino отдельную учетную запись в Twitter.

1. Посетите сайт http://twitter.com/ и создайте учетную запись для платы Arduino. Запишите имя пользователя и пароль.
2. Получите ключ от стороннего сайта http://arduino-tweet.appspot.com/, который создаст мост между вашей платой Arduino и службой Twitter. Это может быть сделано всего за один шаг.
3. Скопируйте и вставьте ключ (вместе с информацией о новой учетной записи Twitter для Arduino) в текстовый файл на вашем компьютере.
4. Загрузите и установите библиотеку Twitter Arduino: https://github.com/NeoCat/Arduno-Twitter-library/archive/master.zip.

Оборудование

Ниже перечислено оборудование для нашего проекта:

- один кабель USB;
- один сетевой кабель;
- одна плата Arduino Uno с платой расширения Ethernet или одна плата Freetronic EtherTen.

Скетч

Введите следующий скетч, *но пока не загружайте его*:

```
// Проект 61 – Arduino Tweeter
#include <SPI.h>
#include <Ethernet.h>
#include <Twitter.h>

❶ IPAddress ip(192,168,0,1); // Замените своим IP-адресом
❷ byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
❸ Twitter twitter("вставьте_сюда_свой_ключ");
```

```
  // Сообщение для отправки
❹ char msg[] = "I'm alive!";

  void setup()
  {
    delay(1000);
    Ethernet.begin(mac, ip);
    // Или, если IP-адрес настраивается автоматически через DHCP,
    // Ethernet.begin(mac);
    Serial.begin(9600);
    Serial.println("connecting...");
  }

  void loop()
  {
    if (twitter.post(msg)) {
      int status = twitter.wait();
❺     if (status == 200) {
        Serial.println("OK.");
      } else {
        Serial.print("failed : code ");
        Serial.println(status);
      }
    } else {
      Serial.println("connection failed.");
    }
    while (1);
  }
```

Как и в проекте 60, вставьте свой IP-адрес в строке ❶ и измените MAC-адрес в строке ❷ при необходимости. Потом вставьте ключ доступа к Twitter в строке ❸, заключив его в двойные кавычки. В строке ❹ вставьте текст, который нужно послать. Теперь загрузите скетч и подключите собранное устройство к сети (не забудьте подписаться на получение сообщений, созданных под учетной записью Arduino!). Через пару минут откройте свою страницу в Twitter — на ней должно появиться сообщение, как на рис. 21.4.

При создании устройства помните, что посылать сообщения можно не чаще раза в минуту и каждое сообщение должно быть уникальным (это правила Twitter).

В ответ на попытку отправить сообщение Twitter возвращает код результата. Скетч принимает и выводит этот код в монитор порта в строке ❺, как на рис. 21.5.

Если вы получили в ответ код 403, как на рис. 21.5, значит, был указан неверный ключ или вы пытаетесь посылать сообщения слишком часто (полный список кодов ошибок Twitter можно найти по адресу https://finderrorcode.com/twitter-error-codes.html).

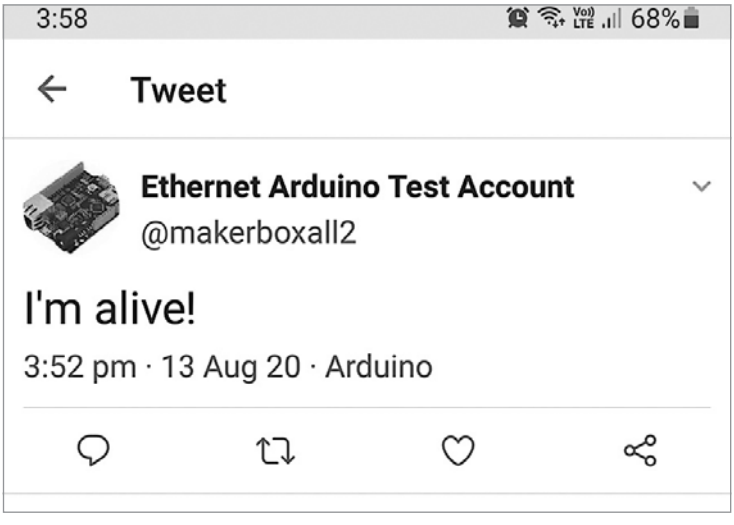


Рис. 21.4. Сообщение, отправленное платой Arduino

```
connecting ...
HTTP/1.1 403 Forbidden
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
X-Cloud-Trace-Context: 31e34e66928b59e94cda735e0944674d;o=1
Date: Thu, 13 Aug 2020 06:02:00 GMT
Server: Google Frontend
Content-Length: 73
Connection: close

Error 403 - {"errors":[{"code":187,"message":"Status is a duplicate."}]}
failed : code 403
```

Рис. 21.5. Результат попытки послать одно и то же сообщение повторно

Управление платой Arduino через интернет

Есть несколько способов управлять платой Arduino через браузер. После некоторых исследований был обнаружен надежный, защищенный и бесплатный метод — Teleduino.

Teleduino — бесплатная служба от Натана Кеннеди (Nathan Kennedy) — энтузиаста Arduino из Новой Зеландии. Это простой, но мощный инструмент организации взаимодействий с Arduino через интернет. Для него не нужно писать сложные или необычные скетчи. Чтобы приступить к управлению платой Arduino, достаточно открыть в браузере страницу со специальным адресом URL. С помощью службы Teleduino можно управлять цифровыми выходами и сервоприводами или

отправлять I²C-команды. Круг ее возможностей постоянно расширяется. В проекте 62 вы узнаете, как настроить службу Teleduino и дистанционно управлять цифровыми выходами с любого устройства, имеющего выход в интернет.

Проект 62: настройка дистанционного управления платой Arduino

Для начала вам нужно зарегистрироваться в службе Teleduino и получить уникальный ключ для идентификации вашей платы Arduino: посетите страницу https://www.teleduino.org/tools/request-key/ и введите требуемую информацию. В ответ вы получите электронное письмо с уникальным ключом, который будет выглядеть примерно так: `187654321Z9AEFF952ABCDEF8534B2BBF`.

После этого преобразуйте ключ в массив, посетив страницу https://www.teleduino.org/tools/arduino-sketch-key/. Введите свой ключ, и он отобразится на странице в виде массива (рис. 21.6).

```
byte key[] = { 0x65, 0x9A, 0xCE, 0xB1,
               0xA7, 0x5E, 0x57, 0x2B,
               0x8F, 0xFE, 0xA4, 0x40,
               0x9E, 0x7B, 0x7A, 0xBC };
```

Рис. 21.6. Ключ Teleduino в виде массива

Каждой плате Arduino соответствует единственный уникальный ключ, но вы можете получить несколько для работы над несколькими проектами на основе Teleduino.

Оборудование

Ниже перечислено оборудование для этого проекта:

- один кабель USB;
- один сетевой кабель;
- одна плата Arduino Uno с платой расширения Ethernet или одна плата, совместимая с Arduino Uno, с интегрированным оборудованием Ethernet;
- один резистор номиналом 560 Ом (R1);
- одна макетная плата;
- один светодиод любого цвета.

Соберите устройство с помощью схемы на рис. 21.7 и подключите светодиод к цифровому контакту 8.

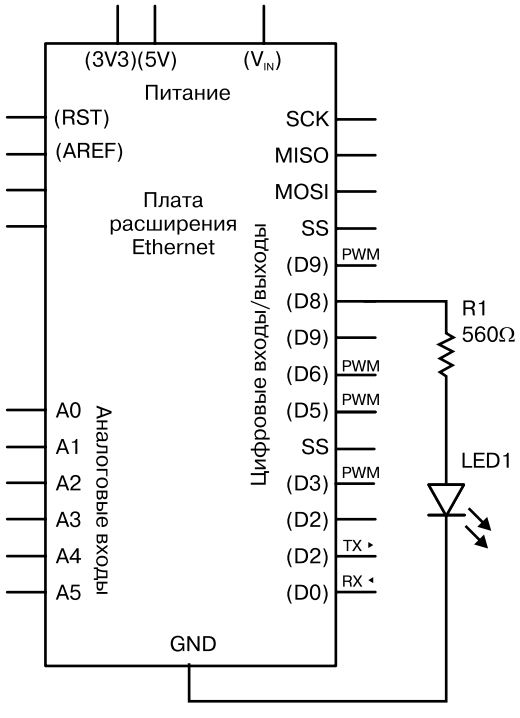


Рис. 21.7. Принципиальная схема проекта 62

Скетч

В проектах на основе Teleduino используется единственный скетч из библиотеки Teleduino. Получить его можно так.

1. Загрузите и установите библиотеку Teleduino: https://www.teleduino.org/downloads/.
2. Перезапустите среду разработки Arduino IDE и выберите в меню пункт File ▸ Examples ▸ Teleduino328 ▸ TeleduinoEthernetClientProxy (Файл ▸ Примеры ▸ Teleduino328 ▸ TeleduinoEthernetClientProxy).
3. Теперь вы должны увидеть скетч Teleduino на экране. Прежде чем загрузить его в плату, замените ключ по умолчанию массивом из своего ключа. Переменная для замены находится в скетче в строке 36. После замены сохраните скетч и загрузите его в Arduino.

Подключите собранное устройство к сети и наблюдайте за светодиодом. Спустя примерно минуту он должен мигнуть несколько раз и погаснуть. Количество вспышек соответствует состоянию службы Teleduino (табл. 21.1).

Таблица 21.1. Коды состояния службы Teleduino

Количество вспышек	Значение
1	Инициализация
2	Открывается сетевое соединение
3	Установлено соединение с сервером Teleduino
4	Аутентификация выполнена успешно
5	Сеанс уже существует
6	Неверный или неавторизованный ключ
10	Соединение закрыто

Если вы увидите пять вспышек, значит, ваш ключ уже запрограммирован в другой плате Arduino, которая сейчас подключена к серверу Teleduino. Увидев десять вспышек, проверьте свое оборудование и подключение к интернету. После того как Arduino установит соединение с сервером, плата должна мигать светодиодом один раз каждые пять секунд. Состоянием светодиода управляет цифровой контакт 8, поэтому его нельзя использовать для других целей, пока плата находится под управлением Teleduino.

Дистанционное управление платой Arduino

Для удаленного управления Arduino с помощью службы Teleduino можно использовать любое устройство с браузером. Но сначала нужно настроить режим для каждого цифрового вывода, которым вы собираетесь управлять. Команда управления посылается вводом созданного вами URL-адреса:

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k=<YOURKEY>&r=definePinMode&pin=
<X>&mode=<Y>
```

Измените три параметра в URL-адресе: замените параметр `<YOURKEY>` алфавитно-цифровым ключом, полученным на сайте Teleduino. Замените `<X>` номером цифрового контакта. Замените `<Y>` значением `1`, соответствующим режиму настройки контакта на работу в роли цифрового выхода.

После этого можно дистанционно управлять состоянием цифрового контакта. Команда управления выглядит так:

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k={YOURKEY}&r=
setDigitalOutput&pin=<X>&output=<S>
```

И снова в ней нужно заменить три параметра: замените параметр <*YOURKEY*> алфавитно-цифровым ключом, полученным на сайте Teleduino. Замените <*X*> номером цифрового контакта. Замените <*S*> значением `0`, соответствующим уровню `LOW`, или 1, соответствующим уровню `HIGH`, для изменения состояния цифрового выхода. Например, чтобы перевести контакт 7 в состояние `HIGH`, нужно ввести адрес:

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k={YOURKEY}&r=
setDigitalOutput&pin=7&output=1
```

В случае успешного выполнения команды вы должны увидеть в окне браузера примерно такой текст:

```
{"status":200,"message":"OK","response"
{"result":0,"time":0.22814512252808,"values":[]}}
```

Если команда потерпит неудачу, в браузере появится примерно такой текст:

```
{"status":403,"message":"Key is offline or invalid.","response":[]}
```

Вы можете посылать команды для изменения состояния цифровых выходов, просто изменяя URL-адрес.

Если цифровой контакт поддерживает возможность работы в режиме широтно-импульсной модуляции (ШИМ), как описано в главе 3, вы сможете управлять выходом ШИМ с помощью команды:

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k={YOURKEY}&r=setPwmOutput&pin=
<X>&output=<Y>
```

где <*X*> — номер цифрового контакта, а <*Y*> — коэффициент заполнения от 0 до 255.

После создания URL-адресов для своего проекта сохраните их в закладках браузера или создайте локальную веб-страницу с важными ссылками в виде кнопок. Можно создать одну закладку с URL-адресом, устанавливающим уровень `HIGH` на цифровом выходе 7, и еще одну — устанавливающую уровень `LOW` на том же выходе.

Иногда состояние выходов на плате Arduino может быть очень важным. Поэтому для безопасности определите состояние по умолчанию цифровых контактов на случай сброса платы Arduino по любым причинам. После подключения устройства к службе Teleduino откройте страницу https://www.teleduino.org/tools/manage-presets/. Введите уникальный ключ, после чего вы увидите массив параметров, в котором можно выбрать режим работы и состояние для каждого цифрового контакта (рис. 21.8).

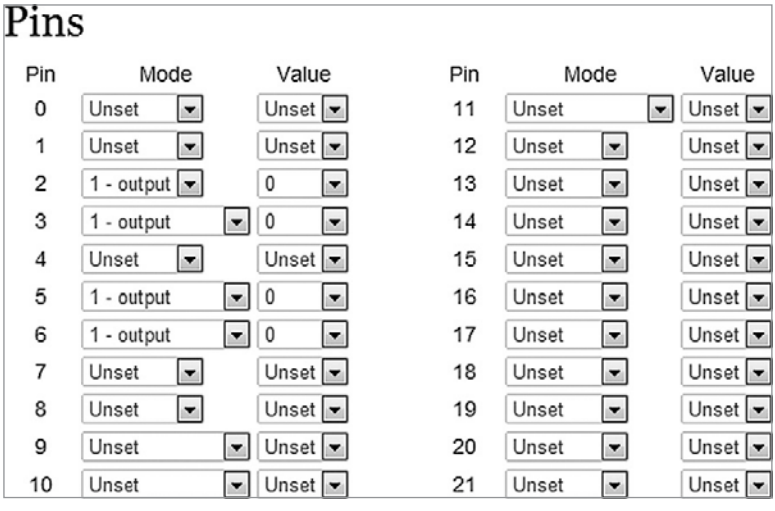


Рис. 21.8. Страница настройки состояния выводов по умолчанию

Что дальше?

Устройства на основе Arduino не только легко поддаются мониторингу через интернет и способны посылать сообщения в Twitter. Ими можно и управлять через интернет. Для этого не нужно создавать сложные скетчи, обладать навыками программирования сетевых взаимодействий или сильно тратиться. При помощи средств дистанционного управления через интернет вы сможете контролировать свои устройства на основе Arduino почти из любой точки мира и расширить их возможности вывода данных. Три проекта из этой главы — фундамент, опираясь на который вы сможете создавать собственные устройства с дистанционным управлением.

Следующая, завершающая глава покажет, как в Arduino отправлять и принимать команды через сети сотовой связи.

22

Сети сотовой связи

В этой главе вы узнаете, как:

- с помощью Arduino набрать телефонный номер при необходимости;
- с помощью Arduino послать текстовое сообщение на мобильный телефон;
- управлять устройствами, подключенными к Arduino, через короткие текстовые сообщения.

Проекты на основе Arduino можно подключать к сетям сотовой связи для обеспечения простого взаимодействия между платой и любым телефоном. Добавив толику воображения, вы сможете придумать массу вариантов применения этих решений, в том числе и приводящихся далее.

Обязательно дочитайте главу до конца перед приобретением любого оборудования. Успех проектов из этой книги во многом зависит от особенностей вашей сети сотовой связи. Она должна:

- поддерживать стандарт: UMTS (3G) 850 МГц, 900 МГц, 1900 МГц или 2100 МГц;
- позволять использовать устройства, не поставляемые оператором связи.

Для проектов ниже лучше использовать тарифные планы с предоплатой или планы, допускающие большое количество текстовых сообщений, на случай если из-за ошибки в скетче ваш проект пошлет несколько коротких СМС. Перед использованием SIM-карты отключите требование ввода PIN-кода. Для этого вставьте SIM-карту в мобильный телефон и в настройках безопасности отключите параметр, требующий ввода PIN-кода.

Оборудование

Во всех проектах используется одинаковый набор оборудования, поэтому сначала настроим его. Для проектов из этой главы понадобится необычное оборудование, начиная с платы расширения 3G GSM типа SIM5320 с антенной (рис. 22.1). Ее можно приобрести в компании TinySine (https://www.tinyosshop.com/) и у дистрибьюторов. Есть две разновидности платы SIM5320: SIM5320A и SIM5320E.

Версия -E использует полосы частот UMTS/HSDPA 900/2100 МГц (в основном для европейских пользователей), а версия -A — полосу частот UMTS/HSDPA 850/1900 МГц (в основном для пользователей из США и Австралии, использующих сеть Telstra).

Рис. 22.1. Плата расширения GSM с подключаемой антенной

Вам понадобится внешний источник питания. Иногда плата расширения 3G потребляет ток до 2 А (что превышает возможности Arduino) и может вывести из строя плату. Предотвратить это поможет любой источник постоянного тока, способный отдавать ток силой до 2 А (например, зарядное устройство для аккумуляторов с выходным напряжением 7.2 В, солнечные панели, батареи, аккумуляторы напряжением 12 В и т. п. с одним условием: выходное напряжение источника постоянного тока не должно превышать 12 В).

Настройка и проверка оборудования

Теперь проверим возможность платы расширения 3G взаимодействовать с сетью сотовой связи и с Arduino. Сначала нужно установить перемычки последовательного порта, так как плата расширения 3G взаимодействует с Arduino через него, как и модули GPS из главы 15. С помощью перемычек вверху справа можно выбрать цифровые контакты, которые будут использоваться для связи с платой Arduino. Во всех проектах из этой главы для связи с платой расширения будут использоваться цифровые контакты 2 (для передачи) и 3 (для приема). Подключите перемычки к контактам TX2 и RX3, как на рис. 22.2.

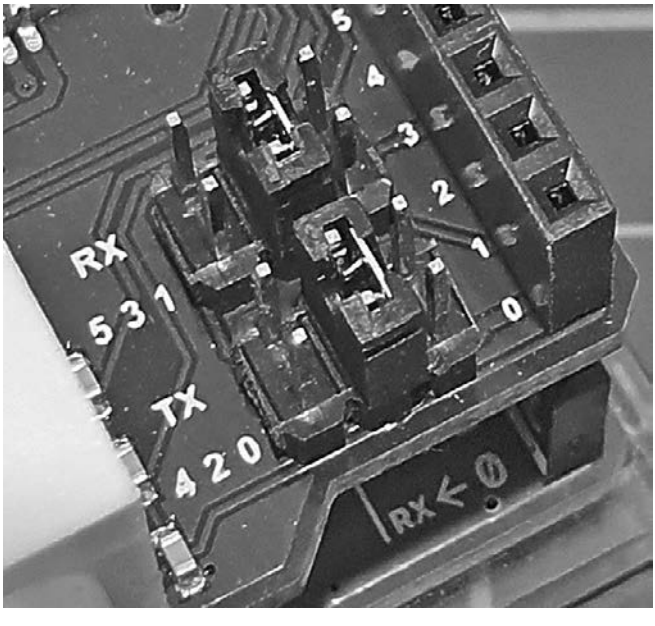


Рис. 22.2. Установка перемычек на плате расширения

Теперь переверните плату расширения и вставьте SIM-карту в держатель, как на рис. 22.3.

После этого аккуратно вставьте плату расширения 3G в разъемы на Arduino. Подключите внешний источник питания и кабель USB к Arduino и ПК. Как и в случае с мобильным телефоном, нужно включить (и выключить) модуль SIM-карты с помощью кнопки питания в верхнем левом углу платы расширения (рис. 22.4). Нажмите кнопку на две секунды и отпустите. Спустя несколько секунд загорятся светодиоды P (power — «питание») и S (status — «состояние»), а синий начнет мигать, как только плата расширения 3G зарегистрируется в сети сотовой связи.

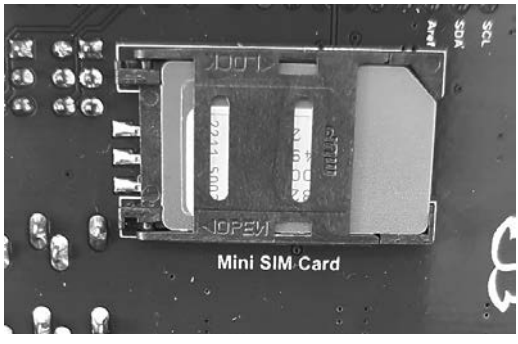


Рис. 22.3. Установка SIM-карты

Запомните, что кнопка питания подключена к цифровому контакту 8, поэтому вы можете управлять питанием из своего скетча вместо включения и выключения вручную.

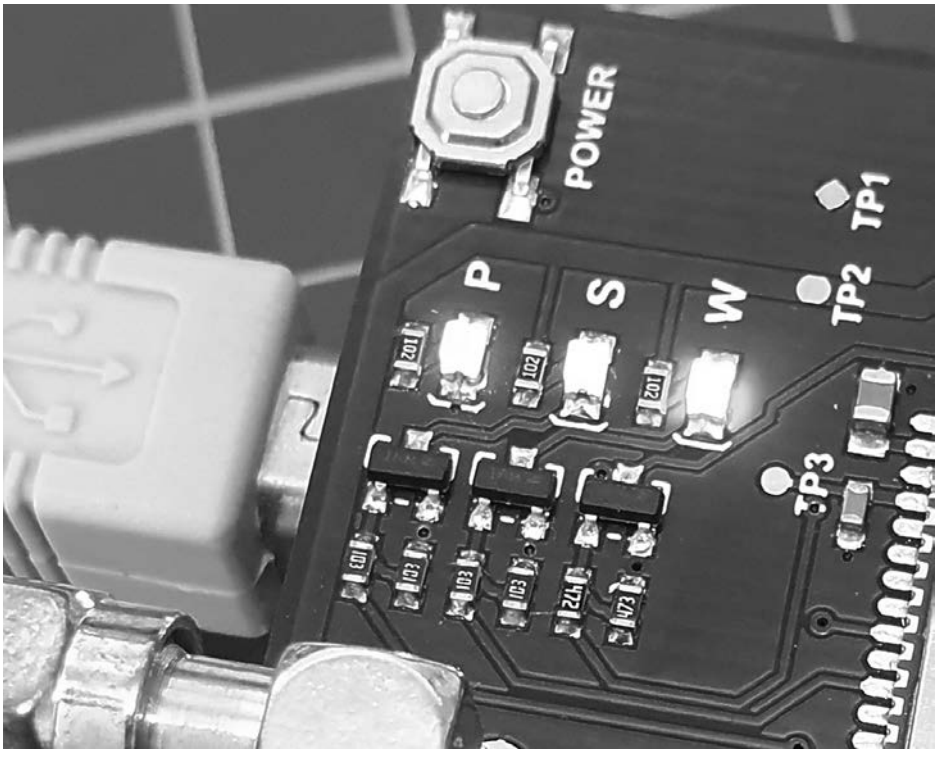


Рис. 22.4. Кнопка включения питания и светодиоды на плате расширения 3G

Теперь введите и загрузите скетч в листинге 22.1.

Листинг 22.1. Скетч для проверки платы расширения 3G

```
❶ #include <SoftwareSerial.h> // Виртуальный последовательный порт
❷ SoftwareSerial cell(2,3);
  char incoming_char = 0;

  void setup()
  {
    // Инициализировать последовательные порты
    Serial.begin(9600);
❸   cell.begin(4800);
    Serial.println("Starting SM5320 Communication...");
  }

  void loop()
  {
    // Если получен символ от модуля 3G...
    if( cell.available() > 0 )
    {
      // Прочитать символ из порта, связанного с модулем 3G
      incoming_char = cell.read();
      // Вывести полученный символ в монитор порта
      Serial.print(incoming_char);
    }
    // Если получен символ из монитора порта...
    if( Serial.available() > 0 )
    {
      incoming_char = Serial.read();// Прочитать символ из монитора порта
      cell.print(incoming_char);    // Послать символ в модуль 3G
    }
  }
```

Этот скетч действует как простое передаточное звено, пересылая всю поступающую от платы расширения GSM информацию в монитор порта. Модуль 3G создает последовательное соединение с Arduino с помощью цифровых контактов 2 и 3, которое не мешает обмену данными по стандартному последовательному порту между Arduino и ПК, подключенному к цифровым контактам 0 и 1. Для связи с платой 3G мы настроили виртуальный последовательный порт (❶, ❷ и ❸). По умолчанию модуль 3G использует скорость обмена через последовательный порт 4800 бод. Ее вполне достаточно для нашего проекта.

После загрузки скетча откройте окно монитора порта и подождите примерно десять секунд. Затем с помощью другого телефона позвоните на номер SIM-карты, установленной в модуле 3G. В окне должны появиться такие данные, как на рис. 22.5.

Плата расширения выводит уведомление `RING` (вызов) после получения входящего вызова и `MISSED_CALL` (пропущенный вызов), когда вызов завершается. Если ваша

сотовая сеть поддерживает идентификацию вызывающего абонента, то спустя короткое время появится номер вызывающего абонента (на рис. 22.5 номер заштрихован по соображениям конфиденциальности). Мы выяснили, что модуль 3G работает. Теперь можно приступать к созданию проектов.

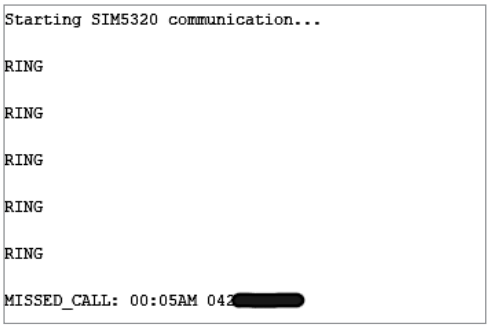


Рис. 22.5. Пример вывода результатов работы скетча из листинга 22.1

Проект 63: автоматический наборщик номера

К концу этого проекта ваша плата Arduino научится набирать телефонный номер по определяемым скетчем событиям. К примеру, если температура в холодильной камере поднялась выше определенного порога или сработала охранная сигнализация, Arduino сможет позвонить по заранее заданному номеру и через 20 секунд повесить трубку. Чтобы определить подобный звонок, занесите номер Arduino в список контактов под понятным вам именем.

Оборудование

В этом проекте используется оборудование, описанное в начале главы, и несколько дополнительных компонентов по вашему выбору и в зависимости от назначения. Для демонстрации мы используем кнопку, нажатие на которую будет вызывать звонок.

В дополнение к описанному выше оборудованию нам понадобятся:

- одна кнопка без фиксации;
- один резистор номиналом 10 кОм;
- один конденсатор емкостью 100 нФ;
- несколько отрезков провода разной длины;
- одна макетная плата.

Схема

Подключите дополнительные компоненты как на рис. 22.6.

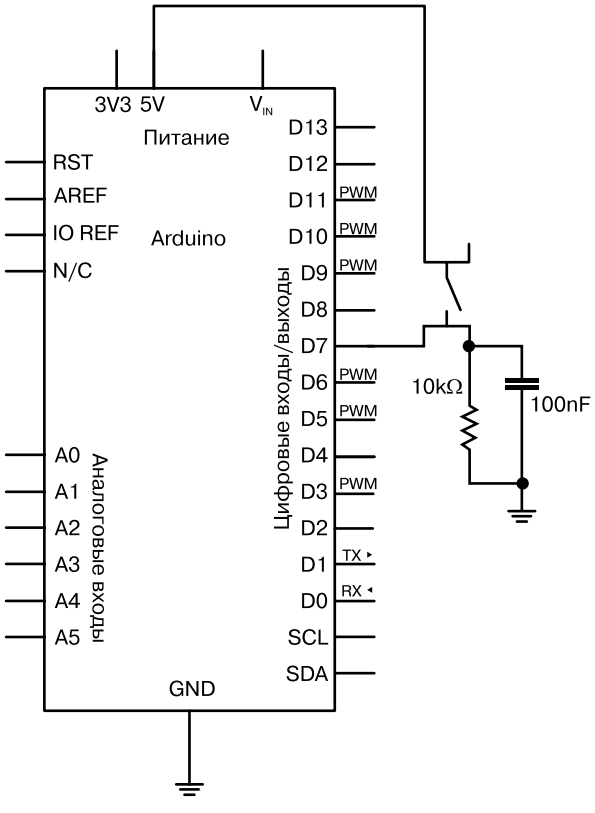


Рис. 22.6. Принципиальная схема проекта 63

Скетч

Введите, *но пока не загружайте* следующий скетч:

```
// Проект 63 – автоматический наборщик номера

#include <SoftwareSerial.h>
SoftwareSerial cell(2,3);
char incoming_char = 0;

void setup()
{
```

```
  pinMode(7, INPUT);  // Кнопка
  pinMode(8, OUTPUT); // Управление питанием модуля 3G
  // Инициализировать последовательный порт для взаимодействий
  Serial.begin(9600);
  cell.begin(4800);
}

void callSomeone()
{
  // Включить модуль 3G
❶ Serial.println("Turning shield power on...");
  digitalWrite(8, HIGH);
  delay(2000);
  digitalWrite(8, LOW);
  delay(10000);
❷ cell.println("ATDxxxxxxxxxx"); // Набрать номер xxxxxxxxxx
  // Замените xxxxxxxxxx желаемым номером телефона (с кодом страны)
  Serial.println("Calling...");
  delay(20000);                 // Ждать 20 секунд
❸ cell.println("ATH");          // Повесить трубку
  Serial.println("Ending call, shield power off.");
  // Выключить модуль 3G для экономии электроэнергии
❹ digitalWrite(8, HIGH);
  delay(2000);
  digitalWrite(8, LOW);
}

void loop()
{
❺ if (digitalRead(7) == HIGH)
  {
❻   callSomeone();
  }
}
```

Принцип действия

После настройки программного и обычного последовательного портов скетч ждет нажатия кнопки, подключенной к контакту 7 ❺. В момент нажатия кнопки вызывается функция `callSomeone()` ❻. Она переводит цифровой вывод 8 в состояние `HIGH` на две секунды, чтобы включить модуль 3G, потом ждет десять секунд, пока модуль зарегистрируется в сотовой сети. Затем скетч посылает команду набора номера ❷. По завершении звонка ❸ питание платы расширения выключается ❹.

Замените строку *xxxxxxxxxx* номером телефона, на который будет звонить плата Arduino. Он должен выглядеть так же, как на обычном мобильном телефоне.

Например, если вы хотите, чтобы Arduino звонила на номер 212-555-12-12, измените строку ❷ в скетче так:

```
cell.println("ATD2125551212");
```

После ввода номера телефона загрузите скетч, подождите минуту, чтобы дать модулю 3G время зарегистрироваться в сотовой сети, и нажмите кнопку. Функцию автоматического набора номера можно интегрировать в любой скетч, потому что ее легко вызвать в нужный момент ❷. Теперь вам остается только найти причины, по которым плата Arduino должна позвонить на ваш номер.

Сейчас перенесем вашу Arduino в XXI век, научив ее посылать текстовые сообщения.

Проект 64: отправка текстовых сообщений

В этом проекте плата Arduino будет по событию посылать текстовое сообщение на другой мобильный телефон. Чтобы упростить скетч, мы будем использовать библиотеку SerialGSM: https://github.com/meirm/SerialGSM/archive/master.zip. После ее установки перезапустите Arduino IDE.

Для этого проекта нужно то же оборудование, что и для проекта 63.

Скетч

Введите следующий скетч в Arduino IDE, но *пока не загружайте его*:

```
// Проект 64 – отправка текстовых сообщений

#include <SerialGSM.h>
#include <SoftwareSerial.h> // Виртуальный последовательный порт
❶ SerialGSM cell(2,3);

void sendSMS()
{
❷   cell.Message("The button has been pressed!");
  cell.SendSMS();
}

void setup()
{
  pinMode(7, INPUT);  // Кнопка
  pinMode(8, OUTPUT); // Управление питанием модуля 3G
```

```
    // Включить модуль 3G
    Serial.println("Turning shield power on...");
    digitalWrite(8, HIGH);

    delay(2000);
    digitalWrite(8, LOW);
    // Инициализировать последовательный порт для взаимодействий
    Serial.begin(9600);
    cell.begin(4800);
    cell.Verbose(true);
    cell.Boot();
    cell.FwdSMS2Serial();
❸   cell.Rcpt("xxxxxxxxxx");
    delay(10000);
  }

  void loop()
  {
❹   if (digitalRead(7) == HIGH)
    {
      sendSMS();
    }
  }
```

Принцип действия

Модуль 3G настраивается в функции `void setup()` как обычно ❶. В строке ❹ определяется нажатие кнопки и вызывается функция `sendSMS()`. Она посылает текстовое сообщение на телефон, номер которого указан в строке ❸.

Прежде чем загрузить скетч, замените строку *xxxxxxxxxx* номером телефона получателя с кодом региона: код региона и номер без пробелов и скобок. Например, чтобы отправить текстовое сообщение на телефон с номером 212-555-12-12 в США, введите номер `2125551212`.

Функция посылает текстовое сообщение, хранящееся в строке ❷ (обратите внимание, что максимальный размер сообщения — 160 символов)[1].

Определив свое текстовое сообщение и номер телефона получателя, загрузите скетч, подождите 30 секунд и нажмите кнопку. Через несколько мгновений сообщение должно быть принято на телефоне получателя (рис. 22.7).

Код из проекта 64 легко можно встроить в любые скетчи и посылать с его помощью разные текстовые сообщения, реализовывая выбор инструкцией `switch-case`.

---

[1] Длина сообщения, написанного кириллицей, не может превышать 70 символов. — *Примеч. пер.*

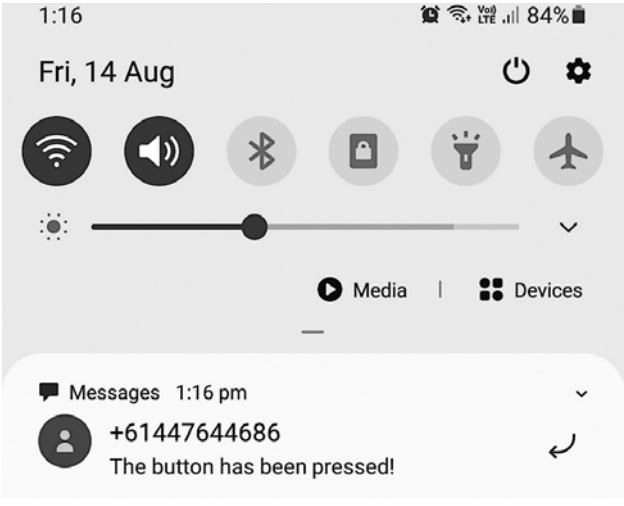


Рис. 22.7. Пример полученного текстового сообщения

ПРИМЕЧАНИЕ

Не забывайте, что отправка текстовых сообщений не бесплатная. Поэтому для экспериментов лучше использовать SIM-карту с тарифным планом без ограничений или с большим количеством предоплаченных сообщений.

Проект 65: дистанционное управление устройствами через короткие текстовые сообщения

В этом проекте мы будем управлять цифровыми выходами на плате Arduino, посылая текстовые сообщения с телефона. У вас уже достаточно знаний, чтобы управлять разными устройствами. В этом проекте будет осуществляться управление четырьмя цифровыми выходами, но вы можете задействовать любое количество.

Чтобы включить или выключить один из четырех цифровых выходов (в нашем примере — контакты с 10 по 13), нужно отправить на номер Arduino текстовое сообщение: `#axbxcxdx`, заменив *x* цифрой `0` или `1`. Например, чтобы установить высокий уровень на всех четырех выходах, отправьте сообщение `#a1b1c1d1`.

Оборудование

В этом проекте используется оборудование из начала главы плюс дополнительные компоненты по выбору. Для индикации состояния управляемых цифровых выходов будут использоваться четыре светодиода.

Для проекта нужно следующее оборудование:

- четыре светодиода;
- четыре резистора номиналом 560 Ом;
- несколько отрезков провода разной длины;
- одна макетная плата.

Схема

Подключите дополнительные компоненты как на рис. 22.8.

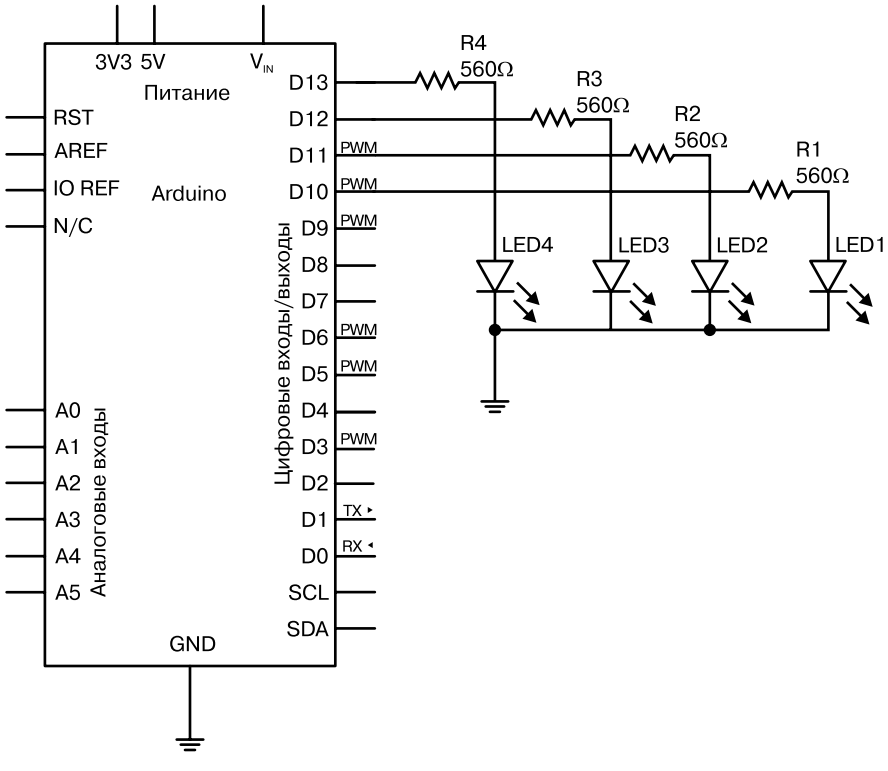


Рис. 22.8. Принципиальная схема проекта 65

Скетч

В этом проекте не будет использоваться библиотека поддержки модуля 3G. Все управление будет производиться с использованием низкоуровневых команд. Скетч не будет включать и выключать модуль 3G, поэтому его потребуется включать

вручную, когда он должен принимать входящие текстовые сообщения. Введите и загрузите следующий скетч:

```
// Проект 65 — дистанционное управление устройствами через
//             короткие текстовые сообщения

#include <SoftwareSerial.h>
SoftwareSerial cell(2,3);
char inchar;

void setup()
{
  // Настроить управляемые цифровые выходы
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);

  digitalWrite(10, LOW); // Состояние выходов по умолчанию
  digitalWrite(11, LOW); // после включения питания или сброса
  digitalWrite(12, LOW); // Измените в соответствии со своими
  digitalWrite(13, LOW); // потребностями

  // Инициализировать последовательный порт связи с модулем GSM
  cell.begin(9600);
  delay(30000);
❶ cell.println("AT+CMGF=1");
  delay(200);
❷ cell.println("AT+CNMI=3,3,0,0");
  delay(200);
}

void loop()
{
  // Если получен символ от модуля GSM...
❸ if(cell.available() > 0)
  {
    inchar = cell.read();
❹   if (inchar == '#') // Начало команды
    {
      delay(10);
      inchar = cell.read();
❺     if (inchar == 'a')
      {
        delay(10);
        inchar = cell.read();
        if (inchar == '0')
        {
          digitalWrite(10, LOW);
        }
        else if (inchar == '1')
```

```
        {
          digitalWrite(10, HIGH);
        }
        delay(10);
        inchar = cell.read();
        if (inchar == 'b')
        {
          inchar = cell.read();
          if (inchar == '0')
          {
            digitalWrite(11, LOW);
          }
          else if (inchar == '1')
          {
            digitalWrite(11, HIGH);
          }
          delay(10);
          inchar = cell.read();
          if (inchar == 'c')
          {
            inchar = cell.read();
            if (inchar == '0')
            {
              digitalWrite(12, LOW);
            }
            else if (inchar == '1')
            {
              digitalWrite(12, HIGH);
            }
            delay(10);
            inchar = cell.read();
            if (inchar == 'd')
            {
              delay(10);
              inchar = cell.read();
              if (inchar == '0')
              {
                digitalWrite(13, LOW);
              }
              else if (inchar == '1')
              {
                digitalWrite(13, HIGH);
              }
              delay(10);
            }
          }
          cell.println("AT+CMGD=1,4"); // Удалить все СМС
        }
      }
    }
  }
}
```

Принцип действия

В этом проекте плата Arduino проверяет каждый символ в текстовом сообщении, полученном от модуля GSM. В строке ❶ плате расширения передается команда преобразовать входящее CMC в текст и послать его содержимое в виртуальный последовательный порт ❷. Потом Arduino просто ждет получения сообщения от модуля GSM ❸.

Поскольку команды управления цифровыми выходами, которые посылаются с телефона и передаются модулем 3G в Arduino, начинаются с символа `#`, скетч ждет его появления в текстовом сообщении ❹. В строке ❺ проверяется первый параметр `a` — если за ним следует `0` или `1`, на выходе устанавливается низкий или высокий уровень соответственно. Процесс повторяется для следующих трех выходов, управляемых символами `b`, `c` и `d`.

Представьте, как просто с помощью этого проекта можно управлять индикаторами, электродвигателями, звонками и многими другими устройствами.

Что дальше?

Реализация трех проектов из этой главы позволила вам заложить прочный фундамент, на котором можно строить свои проекты, поддерживающие взаимодействия с применением сетей сотовой связи. Ваши возможности ограничены только вашим воображением. Можно организовать посылку текстового сообщения в случае затопления подвала или включать кондиционер со своего телефона. Напомню еще раз: не забудьте положить достаточно средств на счет, прежде чем запустить проект в работу.

Сейчас, после знакомства с 65 проектами (и, надеюсь, их воплощения) из этой книги, вы уже обладаете достаточным количеством знаний, чтобы уверенно конструировать свои устройства на основе Arduino. Вы познакомились с основными строительными блоками, составляющими большинство проектов, и я верю, что с помощью описанных приемов вы сможете решить любые задачи и получить от этого удовольствие.

Я буду рад обсудить проекты из этой книги и получить ваши отзывы и пожелания на моем веб-форуме по адресу https://nostarch.com/arduino-workshop-2nd-edition/.

И помните — это только начало. Еще больше информации о самых разных устройствах с идеями и рекомендациями по их использованию вы найдете в сообществе пользователей Arduino в интернете (например, на сайте Arduino: http://forum.arduino.cc/[1]) или в местных клубах.

Не сидите на месте — действуйте!

[1] Есть и русскоязычный сайт пользователей Arduino: http://arduino.ru/. — *Примеч. пер.*

Джон Бокселл

Изучаем Arduino. 65 проектов своими руками

2-е издание

Перевел с английского *А. Киселев*

Руководитель дивизиона	*Ю. Сергиенко*
Ведущий редактор	*Н. Гринчик*
Научный редактор	*В. Трондин*
Литературный редактор	*Т. Сажина*
Художественный редактор	*В. Мостипан*
Корректоры	*М. Молчанова, Е. Павлович*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 05.2022. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 25.03.22. Формат 70×100/16. Бумага офсетная. Усл. п. л. 36,120. Тираж 1000. Заказ 0000.